

ALL-IN-1 Information Update
May 1992

Part Number AA-ND93C-BE

ALL-IN-1 Information Update

Published by
Corporate User Information Products (MRO1-3/C9)
Digital Equipment Corporation
P. O. Box 1001
Marlborough, MA 01752-9840

The *ALL-IN-1 Information Update* is a quarterly publication reporting on important ALL-IN-1 Software Performance Reports (SPRs), management, and tuning. It also describes workarounds and provides customers with information that may help them avoid problems. Much of the material is developed from SPR answers significant to the general audience and is printed here to supplement the maintenance updates.

Distribution

The *ALL-IN-1 Information Update* is directed to one software contact for each software product. No mailing will be made to addresses without a software contact name. **Address change requests should be sent to the nearest Digital field office. Include the new address and mailing label from the most recently received publication.**

The prerequisite to using Digital Software is the appropriate license. The standard Terms and Conditions and Digital Business Agreement (DBA) contain the license terms for all software other than DECsystem-10.

Susan A. Thompson, Editor

Copyright © Digital Equipment Corporation 1992. All Rights Reserved.

The material in this document is for informational purposes only. Digital believes the information in this publication is accurate as of its publication date; such information is subject to change without notice. Digital is not responsible for any inadvertent errors. Comments on the contents of this publication should be directed to your local Digital field office.

The following are trademarks of Digital Equipment Corporation: ALL-IN-1, CDA, DECdecision, DECnet, DECsystem, DECtrace, DECUS, DECUS logo, DECwrite, Digital, FMS, RA, ULTRIX, VAX Notes, VAXcluster, VMS, WPS-PLUS, and the DIGITAL logo.

The following are third-party trademarks: PostScript is a registered trademark of Adobe Systems, Inc.

CONTENTS

	Page
INTRODUCTION	1
USING ALL-IN-1	
Hints for Users	3
User-Defined Processes	7
MANAGEMENT	
ALL-IN-1 System Monitoring	11
Monitoring Electronic Messaging	13
Checking That VMS Setup Is Appropriate for Your ALL-IN-1 System	21
Ensuring the Integrity of an ALL-IN-1 System	23
Identifying Users Logged in to Particular ALL-IN-1 Accounts	25
SYSTEM MANAGEMENT	
Managing an Application for Performance	27
Problem in X.400 Addresses with Multiple Organizational Units	31
APPLICATION PROGRAMMING	
Programming Hints: Working with Symbols	35
Programming Hints: Date and Time Fields	39
Programming an Application for Performance	45
Programming for Optimal Field Access	53
Working with Compound Documents in ALL-IN-1	57
Using the .FUNCTION and .FX Script Directives in ALL-IN-1 Scripts	71
Phantom Data Sets	75
Cumulative Index	79
Attention U.S. Software Contract Customers Submitting Software Problems	87
Digital Equipment Computer Users Society (DECUS)	89
DECUS Subscription Service	93
Reader's Comments	95

INTRODUCTION

Welcome to the May 1992 issue of the *ALL-IN-1 Information Update*. The *ALL-IN-1 Information Update* is a quarterly publication that aims to keep you up-to-date with ALL-IN-1 and contains useful information for system managers, application programmers, and ALL-IN-1 users.

This is a special issue because it includes every article from the last six issues of the *ALL-IN-1 Information Update* that applies to ALL-IN-1 Version 3.0, that is, since August 1990. Where necessary, these articles have been updated. Therefore, if you have installed ALL-IN-1 Version 3.0, the information in this issue supersedes information in earlier issues of the *ALL-IN-1 Information Update*.

Some of the articles published in the *ALL-IN-1 Information Update* are developed from reports of problems that we have received from customers and are included to help you to avoid these problems. Other articles contain practical information about ALL-IN-1 features of which users might not be aware.

If you have any questions about ALL-IN-1 Versions 2.3, 2.4, or 3.0 and are covered by a service contract, contact your local Digital Customer Support Center (CSC).

We are eager to provide you with helpful information. Therefore, please take the time to complete the Reader's Comments page at the back of this issue. This will help us get a better idea about the type of articles you would like to see in future issues.

USING ALL-IN-1

Hints for Users

WPS-PLUS: Making GOLD GET Work More Quickly

When you incorporate text from one document into another document that you are writing, you use GOLD GET. By default, the lines of the text appear on the screen as they are incorporated. To merge the documents more quickly, you can switch the display feature off in the following way:

1. When you are writing or editing a WPS-PLUS document, press GOLD MENU.
2. When the Editor menu is displayed, enter EA. The Editor Attributes form appears.
3. Tab down to the GET DOCMT screen on field.
4. Enter N, since you do not want to see the document scroll as it is copied.
5. Press RETURN.
6. Press EXIT to return to your document.

The next time you use GOLD GET, the document will not be scrolled on the screen while it is being incorporated.

When you switch off the scrolling feature for GOLD GET in one document, it remains switched off in all documents until you change the setting again.

Reading Mail: Skipping Long Distribution Lists

When you read a mail message on the screen beginning with a long distribution list, press GOLD DOWN ARROW once to move to the beginning of the message text.

Printing Mail: Omitting the Distribution Lists

When you want to print a message without the distribution list, use the File Text (FT) option to file the message without the mail header and then print the filed document.

If the message was created as an attachment (identified by a preceding dotted line and the word ATTACHMENT), use the File Attachment (FA) option to file the message only and then print the filed document.

Clearing Out Old Mail

Read and Outbox Folders

Documents in the Read and Outbox folders take up valuable disk space. Unlike the Wastebasket, the Read and Outbox folders are not emptied by the ALL-IN-1 housekeeping procedure. You should, therefore, regularly place unwanted Read messages in the Wastebasket by the Delete option and take the following steps to control the size of your Outbox folder:

- Check your Outbox regularly to find how many mail messages you created and the date you created them. To do this, enter IO from the Electronic Messaging (EM) menu to select the index of your Outbox. Press GOLD BOTTOM to give the name and date of the oldest message at the bottom of the list.
- To delete all the messages in the Outbox, enter ALL and press RETURN to select all the messages. An x before each message shows the message is selected. Enter XD and all the selected messages are deleted.
- You may not want to delete all the messages. To keep the ones you want:
 1. Select all the messages using ALL.
 2. Use the UP or DOWN ARROW to move the cursor to a message you want to keep.
 3. Press GOLD SELECT. This removes the x prefix. Do this with all the messages you want to keep.
 4. Enter XD.
 5. Press RETURN. All the messages will be deleted except the ones you want to keep.

Other Folders

If, after reading mail, you regularly use the Refile document (RFD) option to file the Read messages into other folders, remember the messages remain on the disk, continuing to take up valuable space. Many of these refiled messages may be out-of-date or redundant. To find out where and when you refiled old mail messages, do the following:

- From the Electronic Messaging (EM) menu, enter Index (I).
- Move the cursor to the previous field and delete the name of the folder.
- Press TAB to move the cursor to the TO: field after the CREATION field.
- Enter a date in the TO: field, for example 10-JAN-90, for a list of all mail refiled before that date.
- Press RETURN.

The list gives you the date of each message and the name of the folder where it is refiled.

To delete any unwanted documents, use the procedure described above for the Read and Outbox folders.

Saving Paper When Printing

When you print a document, you can avoid extra flag pages and burst pages by altering your user setup. To do this, from the Main menu enter More menus (M). Then enter User setup (US) and Print options (PO) from the subsequent menus. Check your settings on your user setup. Make sure all the settings are set to N. This eliminates a number of unnecessary pages.

User-Defined Processes

Introduction

This article describes five user-defined processes (UDPs). A UDP contains a set of keystrokes that you can define and store for future use. Once you store the keystrokes in a UDP, you can use the UDP whenever you want to perform that set of keystrokes. You write a UDP in the same way that you write a document.

To create and maintain UDPs, use the User-Defined Process Maintenance menu. To display this menu, enter WP UD.

To invoke a UDP, press DO, then choose the UDP you want to use by entering its name. In the following descriptions, a name for each UDP is suggested. The name is also annotated in the UDP and prefixed by an exclamation mark (!), which shows it is a comment for reference.

A complete description of creating, editing, and using UDPs is in the chapter on user-defined processes of the *ALL-IN-1 Users' Reference*.

Finding Large Private Documents

ALL-IN-1 users can use up much disk space storing private documents in their File Cabinets.

An electronic mail message is only a private document until it is sent: when you send a message to other ALL-IN-1 users on the same system, ALL-IN-1 makes a single copy of the message in a shared area and allows all addressees of the message to access that copy. A word-processing document, on the other hand, is always a private document: if you send or forward it, ALL-IN-1 makes another copy in a shared area (which is accessed by all addressees), but retains the original private copy in your File Cabinet.

You can reduce the amount of disk space you use by deleting private copies of documents that are no longer needed. Obviously, it is best to delete large documents. The following UDP finds all the private documents in your File Cabinet and lists them in descending order of size in your Scratch Pad.

A suggestion for the name of this UDP is F_PD.

```
!F_PD
.FX DISPLAY Finding private documents...
.FX FORCE
.FX GET OA$DCL="OPEN/WRITE TEMP TEMP.TMP"
.FX FOR CAB$ WITH .DA$POINTER = "P" DO -
    GET OA$DCL='X=F$FILE_ATTRIBUTES(" .FILENAME "',"ALQ")' \\ -
    GET OA$DCL="IF F$LENGTH(X) .EQS. 3 THEN Y="''X'"" \\ -
    GET OA$DCL="IF F$LENGTH(X) .EQS. 2 THEN Y="0''X'"" \\ -
    GET OA$DCL="IF F$LENGTH(X) .EQS. 1 THEN Y="00''X'"" \\ -
    GET OA$DCL="Z="''Y' " .CREATED:12 .FOLDER:15 ' ' -
    .DOCNUM:7 .TITLE """" \\ -
    GET OA$DCL="WRITE TEMP ""''Z'"" "
.FX GET OA$DCL="CLOSE TEMP"
.FX PURGE TEMP.TMP
.FX DISPLAY Sorting private documents into size order...
.FX FORCE
.FX GET OA$DCL="SORT TEMP.TMP PASTE.TMP/KEY=(POS:1,SIZE:6,DESCENDING)"
```


USING ALL-IN-1

```
.FX PURGE PASTE.TMP
.FX DISPLAY The list is in your Scratch Pad
.FX LIST PASTE.TMP
```

The UDP displays the folder and document number of each private document. To find out whether you still want that document, select it at a menu and read it. If you no longer require the document, delete it. Once you delete all the private documents you no longer require, use the Empty wastebasket (EW) option to remove them from your account.

Changing the Address List Part Way Through Writing a Message

When you are writing a message, and especially when you are answering a message, you may remember that you need to include someone else in the list of addressees or that you need to change the address list in some way. This UDP lets you edit the address list and the header of the current message.

A suggestion for the name of this UDP is EM_MH.

```
!EM_MH
!      Edits the addressee list and header of the current mail message
!
.FX MAIL EDIT/HEADER
{ETB}
```

Checking Who Received Specific Mail

When you are reading mail, you may consider it useful for particular people to see the same mail. You can use a UDP to search quickly all the TO: and CC: addresses of the current message and its attachments to see if any of those people are listed.

The following UDP prompts for a string and searches the TO/CC list of the current message and its attachments for that string.

If it finds just one match, it shows the match. If it finds more than one match, it shows how many and asks you to press GOLD MESSAGE to display the full list.

It then shows whether the message was sent as a TO: or as a CC: and on which attachment the addressee appeared.

A suggestion for the name of this UDP is EM_WHO.

```
!EM_WHO
!      Has X received this message?
!
.FX PROMPT "Enter name to search for: "
.IF OA$PROMPT_DISPOSE EQS 0 THEN .GOTO END
.IF OA$PROMPT_TEXT EQS "" THEN .GOTO END
.FX DECIMAL I
.FX GET #FOUND = 0
.FX FOR CAB$ATTRIBUTES:"TO" WITH .VALUE <=> OA$PROMPT_TEXT DO -
    GET "TO: " .VALUE \\ COMPUTE #FOUND = #FOUND + 1
.FX FOR CAB$ATTRIBUTES:"CC" WITH .VALUE <=> OA$PROMPT_TEXT DO -
    GET "CC: " .VALUE \\ COMPUTE #FOUND = #FOUND + 1
```

```
.FX GET #ATT = 0
.FX FOR CAB$ATTACH DO -
    COMPUTE #ATT = #ATT + 1 \\ -
    FOR CAB$ATTACH_ATTRIBUTES:"TO" WITH .VALUE <=> OA$PROMPT_TEXT DO -
        GET "ATT." #ATT ": TO: " .VALUE \\\ COMPUTE #FOUND = #FOUND + 1

.FX GET #ATT = 0
.FX FOR CAB$ATTACH DO -
    COMPUTE #ATT = #ATT + 1 \\ -
    FOR CAB$ATTACH_ATTRIBUTES:"CC" WITH .VALUE <=> OA$PROMPT_TEXT DO -
        GET "ATT." #ATT ": CC: " .VALUE \\\ COMPUTE #FOUND = #FOUND + 1

.IF #FOUND EQ 1 THEN .GOTO JUST_ONE
.IF #FOUND = 0 THEN .GOTO NONE
.FX GET #FOUND " matches found. Press GOLD MESSAGE for list"
.LABEL END
.FX OA$FLD_STAY
.EXIT

.LABEL JUST_ONE
.EXIT

.LABEL NONE
.FX GET "No matches found for " OA$PROMPT_TEXT
.EXIT
```

Counting the Words in a Document

This UDP allows you to count the number of words in a WPS-PLUS document.

NOTE

This UDP can only be used when working in WPS-PLUS.

A suggestion for the name of this UDP is WP_WORD.

```
!WP_WORD
!+
! Add up the number of words in a document and display
! result at end.
!-
{GOLD T}
.FX DECIMAL I
.FX GET #NUM = 0
.FX GET $OA$MSG_ID = ""
.LABEL START
{WORD}
.IF $OA$MSG_ID EQS "ERRORS WPL_ATBOTTOM" THEN .GOTO END
.FX COMPUTE #NUM = #NUM + 1
.GOTO START
.LABEL END
.TEXT 24,1, 'Total number of words = ' #NUM
```

You can also modify it to count the number of lines, sentences, paragraphs, or pages in a document.

MANAGEMENT

ALL-IN-1 System Monitoring

An inaccurate view of office environments is that the computing resources are needed only from approximately 8:00 a.m. until 6:00 p.m. In fact, users of office environment software, such as ALL-IN-1, increasingly are demanding virtually continuous computer operation. The dependability of office environment applications is vital to the smooth and profitable operation of many businesses.

The following articles outline how to keep an ALL-IN-1 system running with minimal disruptions. It is applicable to ALL-IN-1 Versions 2.3, 2.4, and 3.0. It points out the typical areas of failure so that ALL-IN-1 managers can monitor these areas and prevent problems. The four main areas covered are:

- Monitoring aspects of Electronic Messaging to ensure that mail flows freely
- Monitoring how VMS is set up on the system running ALL-IN-1, to ensure that ALL-IN-1 operates correctly
- Ensuring the integrity of files on the system
- Identifying users logged in to particular ALL-IN-1 accounts

These articles include general hints and ideas about monitoring these areas, but do not give troubleshooting methods or solutions to specific problems.

NOTE

Contact your Digital representative for information about Digital ASSETS Library packages that automate ALL-IN-1 monitoring.

Monitoring Electronic Messaging

This article covers the following:

- Monitoring the sizes of the Sender and Fetcher queues to ensure that mail is flowing freely
- Checking the mail log files to ensure that the Sender and Fetcher are running correctly
- Checking Message Router links to ensure off-node mail is working correctly
- Monitoring the size of the mail areas to ensure they do not get too big

Monitoring the Size of the Sender Queue

The size of the Sender queue on a smooth-running system depends on the amount of mail traffic generated by users. On a large system, the Sender queue may become quite large during peak times, but normally clears when the system is less heavily loaded. First-class local mail does not go through the Sender.

SENDCHECK, which is a program supplied with ALL-IN-1, displays the number of messages waiting in the Sender queue, for example:

```
$ RUN OA$LIB:SENDCHECK
The queue contains 0 messages waiting to be sent.
```

To ensure the regular monitoring of the Sender queue, you can include this command in a command procedure that runs at intervals during the day and mails you the results. If the number of messages waiting to be sent becomes abnormally high, start investigating any cause for failure.

NOTE

Deferred messages on the Sender queue are also counted by the SENDCHECK program, so it is not unusual for a few messages to be listed as *waiting* and not appear to be sent for a number of days.

How to Check for Deferred Messages

To check which messages shown by SENDCHECK are deferred, log in to the ALL-IN-1 manager's account and enter the options MGT MM DQ. This displays a list of the messages on the Sender queue and indicates whether they are deferred, for example:

Messages Waiting to be Sent			
Sender	Posted date	L/R Retries	Subject
System Manager	02-Feb-1990 08:52	D 0	test

The D under the L/R column shows the message is deferred as described in the *ALL-IN-1 Mail Management Guide*.

MANAGEMENT

Recommendations

- Check the Sender queue regularly to ensure mail is flowing freely.
- Create a batch procedure that monitors the Sender queue and that runs at regular intervals (for example, every few hours).
- Check the Sender queue using MGT MM DQ to ensure any messages reported as “Waiting to be Sent” are deferred messages and not caused by the Sender running incorrectly.
- Ensure the Sender is running correctly if the number of messages waiting becomes higher than normal.

Monitoring the Size of the Fetcher Queue

The Fetcher uses two queues:

- The queue of messages held within ALL-IN-1 (in the ALL-IN-1 Fetcher queue)
- The queue of messages waiting in the Message Router ALL-IN-1 mailbox

As with the Sender queue, the number of messages waiting to be fetched varies from system to system, depending on mail traffic. The Fetcher queue can become large at peak times, and clear when the system is less heavily loaded.

How to Check the ALL-IN-1 Fetcher Queue

FETCHCHECK, which is a program supplied with ALL-IN-1, displays the number of messages waiting in the ALL-IN-1 Fetcher queue, for example:

```
$ RUN OA$LIB:FETCHCHECK
The queue contains 0 messages waiting to be fetched.
$
```

The number of messages waiting to be fetched is normally 0 or 1. If the number is greater than 1, check that the Fetcher is running correctly.

How to Check the Message Router ALL-IN-1 Mailbox

Before you can check the queue of messages waiting in the Message Router ALL-IN-1 mailbox, you need to know:

- Which system is running Message Router; this can be the local node or a remote node connected DECnet.

The logical OA\$MTI_MR_NODE contains the name of the Message Router system. If this logical does not exist, Message Router runs on the local node.

- The name of the mailbox ALL-IN-1 uses.

The logical OA\$MTI_MAILBX contains the name of the ALL-IN-1 mailbox. This is often A1.

The following DCL command displays the name:

```
$ SHOW LOGICAL OA$MTI*
```

Enter the following (using your mailbox name instead of mailbox-name) to check the mailbox on the system that is running Message Router:

```
$ RUN SYSS$SYSTEM:MRMAN
MRM> DUMP mailbox-name
```

For example:

```
$ RUN SYSS$SYSTEM:MRMAN
      This is MRMAN V3.2
MRM> DUMP A1
%No queue records
```

This example shows there are no messages waiting in the Message Router ALL-IN-1 mailbox.

```
$ RUN SYSS$SYSTEM:MRMAN
      This is MRMAN V3.2
MRM> DUMP A1

Mailbox file dump for queue A1,

Timestamp      Priority  Serial-Number      Message-id
92031913330844      1        1065      50333191302991/1065@NODEA
92031913332060      1        1066      91333191302991/1066@NODEA
```

This example shows two messages waiting in the Message Router ALL-IN-1 mailbox. The timestamp on the left shows when the messages were posted into the mailbox; if this is a few hours or more before the current time, check that the Fetcher is running correctly. The older the message at the top of the queue, the greater the indication that the Fetcher is not running correctly.

Recommendations

- Use FETCHCHECK to check the ALL-IN-1 Fetcher queue regularly, to ensure mail is flowing freely.
- Check the size of the Message Router ALL-IN-1 mailbox regularly. If the queue becomes abnormally large, check for possible failure.
- Create a procedure that monitors the Message Router ALL-IN-1 mailbox regularly through the day and mails you the results.
- Investigate for problems if either the ALL-IN-1 Fetcher queue or the Message Router ALL-IN-1 mailbox becomes larger than normal.

Checking the Mail Log Files

The ALL-IN-1 Sender and Fetcher produce logs when they run. The log files, OAM-TISEND.LOG and OAMTIMAIL.LOG, respectively, indicate whether or not the Sender and Fetcher are functioning correctly. The ALL-IN-1 system normally keeps the last five generations of these files.

The log files are created in the default login area for the account running the jobs. Normally, this is the ALLIN1 account's directory. Type or edit the log files to ensure there are no errors reported.

MANAGEMENT

Example 1:

```
$ TYPE/PAGE OAMTIMAIL.LOG
.
.
.
$ WRITE SYS$OUTPUT "No messages to fetch"
No messages to fetch
$ IF DEB$LOG THEN WRITE LOGFILE -
" 6-FEB-1990 09:40:07.69      %FETCHER-I-NOMSGS, No messages to fetch from MR"
$ GOTO FIN
$FIN:
$ IF DEB$LOG THEN WRITE LOGFILE -
" 6-FEB-1990 09:40:07.73      %FETCHER-I-END, End of  6-FEB-1990 09:39:58.4
  fetcher run on NODEA"
$ IF DEB$LOG THEN WRITE LOGFILE ""
$ ON ERROR THEN CONTINUE
$ ON WARNING THEN CONTINUE
$ PURGE SYS$LOGIN:OAMTIMAIL.LOG/KEEP=5
$ IF "$255$DUA14" .NES. "" THEN CLOSE LOCKFILE1
$ IF "$255$DUA14" .NES. "" THEN CLOSE LOCKFILE2
$ IF "" .NES. "" THEN CLOSE LOGFILE
ALLIN1      job terminated at  6-FEB-1990 09:40:08.72
```

This run of the Fetcher had no messages to process and completed normally.

Example 2:

```
$ TYPE/PAGE OAMTISEND.LOG
.
.
.
$RUN_SC:
$ !
$ ON ERROR THEN CONTINUE
$ ON WARNING THEN CONTINUE
$ !
$ IF ShutDown .EQS. "ON HOLD" THEN GOTO FIN
$ FIN:
$ IF DEB$LOG THEN WRITE LOGFILE -
"16-FEB-1990 09:12:49.16      %SENDER-I-END, End of 16-FEB-1990 09:12:42.82 se
nder run on NODEA"
$ IF DEB$LOG THEN WRITE LOGFILE ""
$ PURGE SYS$LOGIN:OAMTISEND.LOG/KEEP=5
$ IF "" .NES. "" THEN CLOSE LOCKFILE1
$ IF "" .NES. "" THEN CLOSE LOCKFILE2
$ IF "" .NES. "" THEN CLOSE LOGFILE
ALLIN1      job terminated at 16-FEB-1990 09:12:50.09

Accounting information:
Buffered I/O count:      89      Peak working set size:  552
Direct I/O count:       62      Peak page file size:    3676
Page faults:           831      Mounted volumes:      0
Charged CPU time:      0 00:00:03.06  Elapsed time:      0 00:00:11.14
```

This run of the Sender stopped because the Sender was on hold. Restart the Sender using the options MGT MM SS so the Sender can continue normally.

Example 3:

```

$TY OAMTISEND.LOG
.
.
$SEND:
$ !
$ ! Run A1 to send all messages
$ !
$ IF DEB$LOG THEN WRITE LOGFILE -
"16-FEB-1990 09:33:54.94      %SENDER-I-SEND, Sending messages"
ALLIN1/USER=POSTMASTER/NOINIT
OA$INI_INIT
MAIL MTI_SEND_ALL
%EMD-E-UNRECMBX, mailbox not recognized A1;A1MAILBOX
$ !
$ GOTO FIN
$ FIN:
$ IF DEB$LOG THEN WRITE LOGFILE -
"16-FEB-1990 09:34:10.29      %SENDER-I-END, End of 16-FEB-1990 09:33:48.09
sender run on NODEA"
$ IF DEB$LOG THEN WRITE LOGFILE ""
$ PURGE SYS$LOGIN:OAMTISEND.LOG/KEEP=5
$ IF "___$255$DUA14" .NES. "" THEN CLOSE LOCKFILE1
$ IF "___$255$DUA14" .NES. "" THEN CLOSE LOCKFILE2
$ IF "" .NES. "" THEN CLOSE LOGFILE
    ALLIN1      job terminated at 16-FEB-1990 09:34:11.26

Accounting information:
Buffered I/O count:      242      Peak working set size: 2590
Direct I/O count:       293      Peak page file size: 12571
Page faults:            3237      Mounted volumes:      0
Charged CPU time:      0 00:00:09.31  Elapsed time:      0 00:00:25.77

```

This log shows the Sender failed to connect to Message Router. The error given is “%EMD-E-UNRECMBX, mailbox not recognized A1;A1MAILBOX.” The relevant section of the *ALL-IN-1 Mail Management Guide* describes how to correct this error.

Recommendations

- Check the Sender and Fetcher logs regularly.
- Correct any errors using the methods described in the relevant section of the *ALL-IN-1 Mail Management Guide*.

Checking Message Router Links

ALL-IN-1 uses DECnet to communicate with Message Router, by means of DECnet object 22. If DECnet is unavailable or if the node where Message Router is installed is unavailable for any reason, the Sender and Fetcher cannot function for remote mail. Ensure that Message Router exception reporting is enabled, to inform the Message Router manager of mail problems.

MANAGEMENT

Typically, if Message Router is not available, the following message occurs in the Sender and Fetcher log files:

```
$ OPEN/READ/WRITE/ERROR=OPNLNKERR MRLINK NODEA::"22="
"Error occurred while creating link to message router - message was
%RMS-E-ACC, ACP file access failed"
```

How to Check Message Router is Available on a Local Node

You can check the link to Message Router by making a manual link by means of DECnet object 22. If Message Router is installed on the local node, enter the following at the DIGITAL Command Language (DCL) prompt:

```
$ SHOW LOGICAL MR$NODE
"MR$NODE" = "NODEA::"22=" (LNM$SYSTEM_TABLE)
```

If the logical MR\$NODE does not exist, Message Router was not started correctly. Once this logical exists, create a manual link as follows:

```
$ OPEN/READ/WRITE MRLINK MR$NODE
```

If the dollar sign (\$) returns with no error, Message Router is reachable. To close the manual link, enter:

```
$ CLOSE MRLINK
```

If the OPEN command returns an error, the Message Router manager should investigate further to determine the cause of the problem. To do this, refer to the exception reports and to the *Message Router Management Action Procedures* manual.

A typical error might be as follows:

```
$ OPEN/READ/WRITE MRLINK MR$NODE
%DCL-E-OPENIN, error opening NODEB::"22=" as input
-RMS-E-ACC, ACP file access failed
-SYSTEM-F-LINKEXIT, network partner exited
```

In this example the node NODEB is not running Message Router, so the link cannot be established.

How to Check Message Router on a Remote Node

Check the logical OA\$MTI_MR_NODE to establish which node runs Message Router for ALL-IN-1. This gives the name of the Message Router node.

The logical MR\$NODE should be defined to include this node name. If necessary, redefine MR\$NODE to include this node name. Then attempt to create a manual link, for example:

```
$ DEFINE/EXEC MR$NODE "TEST::"22="
$ OPEN/READ/WRITE MRLINK MR$NODE
%DCL-E-OPENIN, error opening TEST::"22=" as input
-RMS-E-ACC, ACP file access failed
-SYSTEM-F-NOSUCHNODE, remote node is unknown
```

In this example, the node TEST is not known to the local node, which is causing the link to Message Router to fail.

Recommendations

- Ensure Message Router is running correctly. If Message Router is stopped, no remote mail will arrive on the system.
- Use the exception reporting facility within Message Router to check for errors. There is a description of this in the relevant sections of the *Message Router Management Guide*.
- Ensure Message Router is running correctly whenever the Sender or Fetcher queues become large. This helps to determine whether there is a problem with Message Router, rather than the ALL-IN-1 Sender and Fetcher jobs.

Monitoring the Size of the Mail Areas

When ALL-IN-1 users send mail, either locally or remotely, the message is placed in a mail area. These areas are directories on disk that are pointed to by the OA\$SHAR* logicals. If these disks or directories become full, users receive errors when trying to send mail.

Regularly check the disk space available for the mail areas and the number of files in each shared directory. You can either use the Reorganize shared directories (RSD) option and look in the log file produced to see the total space used by the files, or you can use DCL commands.

To find out which disks hold the mail areas, enter the following DCL command:

```
$ SHOW LOGICAL OA$SHAR*
```

This displays the directory specifications of the mail areas, for example:

```
"OA$SHARE" = [ALLIN1.SHARED_E] "
"OA$SHARE1" = "DISK$TECH:[ALLIN1.SHARE1] "
"OA$SHARE10" = "DISK$TECH:[OA$SHARE.SHARE10] "
"OA$SHARE11" = "DISK$TECH:[OA$SHARE.SHARE11] "
"OA$SHARE12" = "DISK$TECH:[OA$SHARE.SHARE12] "
"OA$SHARE13" = "DISK$TECH:[OA$SHARE.SHARE13] "
"OA$SHARE14" = "DISK$TECH:[OA$SHARE.SHARE14] "
"OA$SHARE15" = "DISK$TECH:[OA$SHARE.SHARE15] "
"OA$SHARE16" = "DISK$TECH:[OA$SHARE.SHARE16] "
"OA$SHARE17" = "DISK$TECH:[OA$SHARE.SHARE17] "
"OA$SHARE18" = "DISK$TECH:[OA$SHARE.SHARE18] "
"OA$SHARE19" = "DISK$TECH:[OA$SHARE.SHARE19] "
"OA$SHARE2" = "DISK$TECH:[ALLIN1.SHARE2] "
"OA$SHARE3" = "DISK$TECH:[OA$SHARE.SHARE3] "
"OA$SHARE4" = "DISK$TECH:[OA$SHARE.SHARE4] "
"OA$SHARE5" = "DISK$TECH:[OA$SHARE.SHARE5] "
"OA$SHARE6" = "DISK$TECH:[OA$SHARE.SHARE6] "
"OA$SHARE7" = "DISK$TECH:[OA$SHARE.SHARE7] "
"OA$SHARE8" = "DISK$TECH:[OA$SHARE.SHARE8] "
"OA$SHARE9" = "DISK$TECH:[OA$SHARE.SHARE9] "
"OA$SHARE_HIGH" = " 5"
"OA$SHARE_LOW" = " 1"
```

Use the DCL command `SHOW DEVICE disk name` to see how many free blocks are available for each disk that holds mail areas. Ideally, the number of free blocks should not drop below 5000. For the mail areas shown in the previous example, the following is displayed:

```
$ SHOW DEVICE DISK$TECH
```

Device Name	Device Status	Error Count	Volume Label	Free Blocks	Trans Count	Mnt Cnt
\$255\$DUA14: (DENNIS)	Mounted	0	TECH	131259	225	14

MANAGEMENT

This shows over 100 000 free blocks on the disk.

Problems with performance occur if the number of files in each shared directory becomes too high (over about 500 files in each directory). To check how many files are in each directory, enter the DCL command:

```
$ DIRECTORY/SIZE/TOTAL OA$SHAREynnnn
```

where:

y is a letter from A to E.

nnnn is the number of the mail area, for example:

```
$ DIRECTORY/SIZE/TOTAL OA$SHARE1
Directory DISK$TECH:[ALLIN1.SHARE1]
Total of 31 files, 183 blocks.
```

If the number of files in the directories becomes too high, use the RSD option on the Manage Mail Areas menu to create more mail areas and to change the write window to use the new areas. There is a description of this in the *ALL-IN-1 Mail Management Guide*.

Recommendations

- Do not allow mail areas to have more than about 500 files in each shared directory. Check the number of files in each directory regularly.
- Ensure disks that hold mail areas have at least 5000 free blocks at any one time. Check the free space at least daily, more frequently if space is low.

NOTE

On a large capacity disk like an RA90, there can be plenty of free space, but the maximum number of files allowed may be exceeded.

- Try not to place too many small files on a large capacity disk, but have at least one directory containing large data files (for example, OA\$DAF_%.DAT).
- If disk quotas are enabled on mail area disks, ensure that the account owning the mail areas has sufficient disk quota allocated. This account is typically the ALLIN1 account. Check the disk quota available on a regular basis.

Checking That VMS Setup Is Appropriate for Your ALL-IN-1 System

This article covers the following:

- Checking the modes of logicals used by ALL-IN-1
- Checking the protections of the major files used by ALL-IN-1

Checking the Modes of Logicals

Since Version 2.3, ALL-IN-1 uses the added security feature given by executive mode logicals. Logicals previously defined in supervisor mode must now be defined in executive mode. If the ALL-IN-1 image finds a logical not in executive mode, the image does not raise its privileges to access files, which can prevent users from performing many tasks, including sending mail. Logicals are defined incorrectly if the account starting ALL-IN-1 does not have sufficient privilege to create executive mode logical names.

Use the command `SHOW LOGICAL OA$*/FULL` to show all the ALL-IN-1 logicals. If you are using ALL-IN-1 on a VAXcluster system, remember to check the logicals on all nodes of the VAXcluster, for example:

```
$ SHOW LOGICAL OA$*/FULL
"OA$BLP_BRITISH" [exec] = "DISK$TECH:[ALLIN1.BLP_BRITISH]"
"OA$BLP_LLV" [exec] = "DISK$TECH:[ALLIN1.BLP_BRITISH]"
"OA$BLP_SHARE" [exec] = "DISK$TECH:[ALLIN1.BLP_SHARE]"
"OA$BUILD" [exec] = "OA$SITE_BUILD_LLV"
                  = "OA$SITE_BUILD_SHARE"
                  = "OA$BUILD_SHARE"
"OA$BUILD_SHARE" [exec] = "DISK$TECH:[ALLIN1.SOURCES]"
```

Recommendations

- Always start ALL-IN-1 from a fully privileged account. This ensures logicals are defined correctly in executive mode.
- Check the ALL-IN-1 logicals periodically to ensure they are defined correctly and redefine any that are in supervisor mode.
- To make redefining the logicals easier, create a command file that deassigns all OA\$ logicals defined in supervisor mode and redefines them in executive mode. This ensures that incorrectly defined logicals can be changed quickly and easily.

Checking the Protections on Major Files

ALL-IN-1 data files are protected to ensure data integrity across the system. If the protection is changed, it can cause a security problem or mean users are unable to access the required files.

MANAGEMENT

Use the DCL command DIR/SEC/OWN to check the protections, access control lists (ACLs), and ownership of major data files. The *ALL-IN-1 Management Guide* outlines the recommended protections. Note that the owner of system data files, which is usually ALL-IN-1, should be the controlling account, for example:

```
$ DIR/SEC/OWN OA$SHARE:
Directory DISK$TECH:[ALLIN1.SHARED_E]
OA$DAF_E.DAT;4          [ALLIN1]          (RWED,RWED,,)
```

In this example, the Shared Document Attributes File (SDAF) for mail area E is correctly owned and protected. This ownership and protection is passed to all files in shared directories belonging to that mail area, so it is important that it is correct.

Recommendations

- Regularly check protections and ownership on all major files, as recommended in the relevant section of the *ALL-IN-1 Management Guide*.
- When moving or restoring files, ensure that the protection and ownership are correct before allowing users back on to the system.
- When creating new mail areas, ensure that the protection and ownership are correct before moving users to the new area.

Ensuring the Integrity of an ALL-IN-1 System

This article covers the following:

- Running the Test and Repair housekeeping procedures
- Backing up ALL-IN-1

Testing and Repairing Users' File Cabinets

ALL-IN-1 provides two options to test the integrity of users' File Cabinets and to repair them, if necessary. These options are Test and Repair User Areas (TRU) and Test and Repair Mail Areas (TRM). Both these options are described in the *ALL-IN-1 Management Guide*.

Run these housekeeping procedures on a regular basis, to ensure the integrity of the File Cabinet. If you do not run them frequently, they can take a long time to run: the more frequently you run them, the better the integrity of the File Cabinet, thus, they run more quickly. Also, in ALL-IN-1 Version 2.4 and later, the procedures run faster than previous versions.

If possible, run the procedures weekly (for example, on weekends). If you do not have a heavily used system or if it is not convenient to run them weekly, you can run them monthly.

Backing up ALL-IN-1

Backing up ALL-IN-1 reduces the availability of your system, since you must shut down the system to make the backup. However, it is essential to obtain regular backups, particularly of the SDAFs, which give access to all electronic messages, and so are the hub of the Electronic Messaging subsystem.

The SDAFs are large Record Management Services (RMS) data files, which are regularly accessed by all users. Never back up ALL-IN-1 using the DCL command BACKUP /IGNORE=INTERLOCK, since there is a high risk that a user will access the SDAFs during the backup. In this case, the backup can generate a corrupt copy of an SDAF, even if the SDAF itself is not corrupt. You will not find out that the backup is corrupt until you attempt to recover it later.

If you cannot afford to shut down the system regularly for the time it takes to do a full backup, consider the method described in this section. Note that this method is not a permanent alternative to complete backups, but it can increase the availability of your ALL-IN-1 system.

When you need to do a backup, shut the system down and take a copy of all the SDAFs. Preferably, the copy should be to a large disk with much contiguous empty disk space, but it can be to any disk. When the copy completes correctly, restart ALL-IN-1. You can then back up the SDAFs from the copy to any offline media, such as tape. This backup is not time-critical, since the files are not open while you make the backup.

If you need to restore from a backup made in this way, you must first test the integrity of users' File Cabinets, as described in the *ALL-IN-1 Management Guide*.

Copying the SDAFs to another disk before backing them up saves time because disk-to-disk data transfers are much faster than disk-to-tape data transfers. Your ALL-IN-1 system is, however, still unavailable while the files are being copied because the files must be closed.

MANAGEMENT

The VMS Volume Shadowing product can be used in an enhanced version of this technique. Because shadowed disks are almost always identical copies of one another, the file copy times can be avoided. Use the following steps to back up your SDAFs safely and with only short interruptions in availability. With this method, the SDAFs are unavailable only during steps 2 to 6.

This method requires that the SDAFs reside on shadowed disks.

1. Ascertain that no shadow copy operations are in progress on any shadow set members. If any are active, wait for their completion.¹
2. Shut down ALL-IN-1 in the usual fashion. This closes the SDAFs.
3. Dismount the shadow set. This ensures the updating of on-disk structures.
4. Remove one of the shadow set members.
5. Remount the shadow set with the remaining members.
6. Restart ALL-IN-1 in the usual fashion. Your users can resume their access of the SDAFs at this point.
7. Back up the shadow set member that was removed and contains a consistent copy of the SDAFs.
8. When the backup operation completes, add that member back into the shadow set and let the shadow copy operations *catch up* the disk to current status.

Recommendations

- To avoid periods when the SDAFs are accessed without the protection of shadow set coverage, run with three members in the shadow set. When one member is being backed up, the files still reside on a shadow set with the redundancy of two members.
- To reduce operational errors, use single-disk volumes large enough to hold the SDAF files. Backing up and shadowing bound volume sets is more complex than backing up and shadowing single disk volumes.

¹ The routines \$GETDVI, F\$GETDVI, and LIB\$GETDVI have item-codes DVI\$_SHDW_CATCHUP_COPYING and DVI\$_SHDW_MERGE_COPYING that return TRUE if copy operations are in progress.

Identifying Users Logged in to Particular ALL-IN-1 Accounts

Introduction

This article provides information for the ALL-IN-1 manager about how to determine the users of particular ALL-IN-1 accounts.

ALL-IN-1 normally prevents users from using the same account. It is possible, however, for two or more users to be logged in to the same ALL-IN-1 account and lock the same resources, for example, DAF.DAT, DOCDB.DAT. This can cause ALL-IN-1 to produce unexpected results, for example, a user has problems accessing the File Cabinet because files are locked by the other user.

There are two ways in which a user can log in to an ALL-IN-1 account that is already in use:

- Using /REENTER

A user can log in to ALL-IN-1 specifying the qualifier /REENTER to the ALL-IN-1 command. This can cause problems for both sessions when accessing the File Cabinet.

When a user tries to enter ALL-IN-1 without the /REENTER qualifier (/NOREENTER is the default), ALL-IN-1 does not let the user in. A message indicates ALL-IN-1 is already in use.

- Using NEWDIR

The NEWDIR function (invoked interactively using <NEWDIR) sets the user's default File Cabinet to that of another user, for example:

```
<NEWDIR Vasquez
```

This can cause the same effects as /REENTER does, that is, the user has problems accessing the File Cabinet.

When <NEWDIR is used, the user does not receive an error message if he or she logs into an account that is being used by another user.

Investigating

With problems, it is useful to determine who is locking which resources. This can be achieved with the following DCL command:

```
$ SHOW DEVICE/FILES disk
```

where:

disk is where the user's ALL-IN-1 directory is located, for example, if the DCL command is:

```
$ SHOW DEVICE/FILES DEMO$DUB2:
```


MANAGEMENT

the format of the output is as follows:

Files accessed on device DEMO\$DUB2: on 19-FEB-1990 13:18:11.18

Process name	PID	File name
	00000000	[000000]INDEXF.SYS;1
	00000000	[000000]QUOTA.SYS;1
DEMO1	00001B91	[USER.DEMO2.A1]DOCDB.DAT;27
DEMO1	00001B91	[USER.DEMO2.A1]DAF.DAT;28
DEMO2	00000A24	[USER.DEMO2.A1]USER.FLB;74
DEMO2	00000A24	[USER.DEMO2.A1]DEMO2.PST;2
DEMO1	00001B91	[USER.DEMO2.A1]DEMO2.PST;2
DEMO2	00000A24	[USER.DEMO2.A1]DEMO2.A1CAL;1
DEMO2	00000A24	[USER.DEMO2.A1]NICKNAMES.DAT;4
DEMO2	00000A24	[USER.DEMO2.A1]DOCDB.DAT;27
DEMO2	00000A24	[USER.DEMO2.A1]DAF.DAT;28

As you can see from this output, it is possible to determine which VMS accounts are accessing particular files. For example, user DEMO1 is accessing DOCDB.DAT, DAF.DAT, and DEMO2.PST, which are files in user DEMO2's ALL-IN-1 account.

SYSTEM MANAGEMENT

Managing an Application for Performance

This article gives guidelines on how to manage applications so that they have the best performance, that is, they run as fast as possible and do not slow down the system.

Managing Form and Text Libraries

- Precompile all form libraries (except DEVELOP.FLB).

There is a large overhead associated with accessing forms that are not in a precompiled form library. Forms are kept in a form library (with the .FLB file type), which is a sequential file. When a noncompiled form library is opened, ALL-IN-1 works with the FMS form driver to build a directory of forms in memory so as to access forms as quickly as possible. The first time a noncompiled form is accessed, the named data of the form is compiled and held in memory. From then on, access to that form is as quick as if the form library was precompiled. However, users have already experienced a delay — first when the form library was opened and again when the form was first accessed.

When you display 12 different forms since the last time you accessed a particular form, the first form is dropped from the form cache. If a form is dropped from the form cache, it is compiled again when next accessed.

The time taken to build the directory of forms in memory increases exponentially according to the number of forms in the form library. Another disadvantage with a noncompiled form library is that a dynamically compiled form lasts only while the form is in cache. Once it leaves cache, you need to recompile it again.

When you precompile a form library, the named data of the forms in that library is compiled and held in a compiled form library, which has the same name as the .FLB file and has an .FLC file type. The compiled form library includes a directory of forms: for each form in the compiled form library there is a reference to the address of that form in the uncompiled form library. This ensures that screen data held in the file can be directly accessed without searching through the file.

The development form library (DEVELOP.FLB) is provided as an environment for developing applications. It is not accessible by ordinary users, and its contents are liable to change often. Therefore, you do not need to precompile DEVELOP.FLB.

You compile a form library using the Precompile a form library (PCF) option.

- Install as global sections the compiled form libraries that will be used simultaneously by more than one user.

The cost to the system of installing a form library as a global section in shared memory is small. By installing a form library, users of the forms in the library share the memory used by the library, thus reducing the overall memory requirements of ALL-IN-1. Note, however, that:

- You must not exceed the number of global sections or pages allowed by the system.
- There is little benefit in installing form libraries that are accessed by only one user or are infrequently accessed.

SYSTEM MANAGEMENT

- You must compile the form library before installing it as a global section.
- You compile and, optionally, install a form library using the Precompile a form library (PCF) option.

- Compile and install the text libraries.

DO scripts, SCRIPT scripts, and MERGE templates that are frequently accessed by users should be compiled into a text library (TXL). TXLs are unique to ALL-IN-1. Accessing a file in a TXL provides better performance than accessing the uncompiled version of the file. Once compiled, a TXL should be installed as a global section in shared memory.

You compile and install TXLs from the ALL-IN-1 manager's account. Use the Compile and install CM text library (CCM) option to compile and install the site text library; use the Compile and install ALL-IN-1 text library (CTX) option to compile and install the standard text library.

The advantages of compiling and installing a TXL are:

- Scripts and templates are in shared memory. This reduces the memory needs for the system.
- ALL-IN-1 functions are preparsed, so execution is immediate.

Managing Compute-Intensive Applications

ALL-IN-1 is often used as an interface to compute-intensive applications and batch jobs. It can be useful to run interactive and batch jobs simultaneously in this way, but it does pose some capacity and planning problems. These can usually be solved with a little planning and good knowledge of the applications.

Problems can occur with compute-intensive jobs that:

- Use all the central processing unit (CPU) resources and, hence, increase the response time for ALL-IN-1 users
- Are set at too low a priority and have an unacceptable turnaround time

Possible solutions are:

- Dedicate some portion of the available processing power in a VAXcluster system to serving compute-intensive applications. This requires a business decision on whether the applications are important enough to set aside resources for their sole use.
- Group the applications into categories — trivial, low, medium, and high CPU demands, for instance. Trivial jobs can be run interactively, while low, medium, and high compute-intensive jobs are submitted to the appropriate generic queue and, in turn, to queues on all nodes on the VAXcluster system. Each queue can be set up with the appropriate time limits to force compliance to the categories.

Maintaining Files for Better Performance

As well as the following guidelines, consult the VMS manual *Guide to VMS File Applications* for a full discussion of file maintenance.

- Regularly maintain the data files used by applications.

Both ALL-IN-1 files, such as the File Cabinet, and application files will be affected by maintenance. In the case of ALL-IN-1 files, regular maintenance of files can be controlled by the system manager.

For application files, bear the following in mind:

Record Management Services (RMS) indexed files grow dynamically. They do not shrink dynamically. As index levels grow with the increased number of records, access to records becomes slower and it becomes necessary to maintain the data files.

To maintain files, frequently delete unwanted records and reorganize files. Some ALL-IN-1 housekeeping procedures do automatic reorganizations. Use the DCL CONVERT command for other files. Refer to the documentation on the VMS Convert and Convert/Reclaim Utility.

- Consider the optimum size of buckets when designing an application.

A large data bucket size is better when records are accessed close together on the file. For example, the default data bucket size for DOCDB.DAT is 12. This is because the common user operations that require access to DOCDB, such as Index a folder or Read new mail, often entail reading records that are in the same area on the file as each other or records just accessed.

However, for files where access to records is random, a data bucket size of 2 or 3 is enough. The index bucket size need not be as large as the data bucket size.

For examples of data and index buckets, look at the data files that ALL-IN-1 uses. You can find examples of typical data files in the directories OA\$DATA_SHARE and OA\$DATA_LLIV. Most data files have an equivalent File Definition Language (FDL) file, for example, ACITEM.DAT and ACITEM.FDL. Use ANALYZE/RMS_FILE/STATISTICS at the DIGITAL Command Language (DCL) command level to examine the internal structure of these files or use the many examples of FDL files in OA\$LIB_SHARE as templates for creating your own files.

- Put global buffers on a file if it will be accessed simultaneously by more than one user. Global buffers are more effective when a file is read more than it is written to. A general rule is to have enough global buffers to hold all the index buckets and at least one data bucket.

There is a penalty in having too many global buffers; more CPU resource is needed to search for buffers and, especially in a VAXcluster system, there is more locking activity. On most systems, you should not need more than 400 global buffers on any file.

Problem in X.400 Addresses with Multiple Organizational Units

A problem was discovered when sending messages from or through either of the following two sets of products via the Message Router X.400 Gateway (MRX) to an X.400 recipient whose Originator/Recipient (O/R) name includes more than one organizational unit:

- ALL-IN-1 Electronic Messaging, ALL-IN-1 MAIL
- Message Router/P Gateway (MRP), Message Router/S Gateway (MRS), Message Router Telex Gateway (MR-Telex), ULTRIX Mail Connection (UMC)

The problem occurs when the Message Router Directory Service generates the address of the recipient. The messages from one of the sets of products is rejected by the receiving Message Transfer Agent (MTA). This problem may not affect you now. However, if you introduce one of the other set of products to your mail system or start sending messages to an MTA that is sensitive to the order of organizational units in O/R names, you will experience the problem of messages not being delivered.

If this problem is likely to affect your mail network, please contact your local Customer Support Center (CSC).

For more information about MRX and O/R names, refer to the Message Router X.400 Gateway documentation.

This problem arises because of differences between the two sets of products. The first set does the following:

- ALL-IN-1 Electronic Messaging

ALL-IN-1 uses the Directory Service to supply an X.400 address only when users specifically search the Directory Service database (known in ALL-IN-1 as the Mail Directory) for the address, using the Search mail directory (SMD) option or when ALL-IN-1 is configured to use the Directory Service database primarily.

Users can store the returned address as a nickname to use later. Any addresses obtained and stored in this way are still classed as being supplied by the Directory Service.
- ALL-IN-1 MAIL

ALL-IN-1 MAIL uses the Directory Service to supply an X.400 address only when users specifically search the Directory Service database for the address.

Users can store the returned address in their personal address books to use later. Any addresses obtained in this way and stored in a personal address book are still classed as being supplied by the Directory Service.

The second set of products does this differently, in the following way:

- Message Router/P Gateway

MRP always uses the Directory Service to supply the address of an X.400 recipient.
- Message Router/S Gateway

MRS always uses the Directory Service to supply the address of an X.400 recipient.
- Message Router Telex Gateway

MR-Telex uses the Directory Service to supply the address of an X.400 recipient if it is configured to do so.

SYSTEM MANAGEMENT

- **ULTRIX Mail Connection**

UMC users employ the Directory Service to supply an X.400 address only when they use the address lookup utility that is supplied with UMC.

Users can store the returned address in the alias file to use later. Any addresses obtained and stored in this way are classed as being supplied by the Directory Service.

The two sets of products encode the recipient's O/R name in different ways in the National Bureau of Standards (NBS) form of the message. When MRX receives the NBS form of the message, it translates the message to its X.409 form. During this translation, MRX handles the two different encodings of the O/R name in different ways. Hence, the organizational units in the recipient's O/R name in the X.409 form of messages sent from the two sets of products are in opposite orders of significance. This difference in the order of significance of the organizational units in the recipient's O/R name causes a problem if all three of the following conditions exist:

- You send messages, using products from both sets, to X.400 recipients whose O/R names include multiple organizational units.
- The addresses are supplied by the Directory Service (specifically, the MHS User Identifier field in the Directory Service entry).
- The receiving MTA is sensitive to the order of organizational units in O/R names.

The messages from one of the sets of products will be rejected by the receiving MTA.

The recommended way of setting up Directory Service entries for recipients whose O/R names contain multiple organizational units is to order the organizational units in ascending order of significance in the MHS User Identifier field, that is, from least to most significant. For example, you would use the following MRXMAN command:

```
BUILD SUBSCRIBER /SURNAME=jones
/GIVENNAME=david
/PCOUNTRY=gb
/ADMD=mailnet
/PRMD=bytecorp
/ORGNAME=eurosales
/MHSADDRPART=nodeg
/MHSMAILBOX=b
/MHSUSERID="david jones@cou=gb@admin=mailnet@private
=bytecorp@organization=eurosales@unit=pcsales@unit
=london@unit=southeast@unit=uksales"
```

where:

uksales is the most significant organizational unit.

pcsales is the least significant.

unit is the keyword used to denote the organizational unit.

This gives the following Directory Service entry:

```

ADMDNAME   (Admin Domain)       : MAILNET
PCOUNTRY   (Country Code)       : GB
GIVENNAME  ( First Name(s) )    : DAVID
MRREVLOOKUP (MR Address)        : david jones@cou=gb@admin=mailnet@private
                                     =bytecorp@organization=eurosales@unit=pc
                                     sales@unit=london@unit=southeast@unit=uk
                                     sales@B@NODEG
NAME        (Common Name)       : DAVID JONES
ORGNAME     (Organization Name)  : EUROSALLES
PRMDNAME    (Private Domain Name): BYTECORP
SURNAME     : JONES
MASTER COPY OWNING NODE       : ADDLE
++ ORADDRESS 1
MHSADDRPART (MHS Address)       : NODEG
MHSMAILBOX  (MHS Mailbox)       : B
MHSUSERID   (MHS User Id)       : david jones@cou=gb@admin=mailnet@private
                                     =bytecorp@organization=eurosales@unit=pc
                                     sales@unit=london@unit=southeast@unit=uk
                                     sales

```

APPLICATION PROGRAMMING

Programming Hints: Working with Symbols

This article describes methods for doing the following:

- Adding leading zeros to numbers stored as symbols
- Right-justifying the contents of a symbol in a field or template

The methods described here use the techniques for specifying symbol substrings, described in the chapter on symbols in *Application Programming: Reference Volume 1*.

Adding Leading Zeros

You may want to add leading zeros to a number that is stored as a symbol, for example, a data set whose key contains a numerical value. This is because, if the numbers are of different lengths, ALL-IN-1 does not sort them in numerical order. Instead, it treats the numbers as strings and compares them character by character, starting from the left.

To ensure that symbols containing numbers to be loaded in a field of a form are of equal length, you can use one of the methods described here. Where a field is always entered by a user, you can use the FMS field attributes Right Justify and Zero Fill to ensure that the numbers contain the correct number of characters.

Method 1

This method involves the following steps:

1. Add a large number, such as a power of 10, to the number in the symbol. The number added should end with the same number of zeros as the number of characters that you want the symbol to contain. For example, if the symbol is to fill a six-character field, add 1000000 to the number in the symbol.
2. Specify the length of the symbol substring to match the number of characters you want the symbol to contain and an offset starting position to remove the leading digit that you added in step 1.

For example, if the number in the symbol #NUMBER must fill a field that is six characters long, add 1000000, specify the length of the symbol substring as 6, and specify a starting position of 1 to remove the leading 1, as in the following script:

```
DECIMAL I
COMPUTE #TEMP = #NUMBER + 1000000
GET #NUMBER = #TEMP:6:1
```

You can modify this method to increment or decrement the value of the number held in the symbol. The following example increments the value of #NUMBER by 1 for each iteration of the script:

```
DECIMAL I
COMPUTE #TEMP = #NUMBER + 1000001
GET #NUMBER = #TEMP:6:1
```


APPLICATION PROGRAMMING

The following example decrements the value of #NUMBER by 1 for each iteration of the script:

```
DECIMAL I
COMPUTE #TEMP = #NUMBER + 999999
GET #NUMBER = #TEMP:6:1
```

Method 2

This method involves the following steps:

1. Reverse the number in the symbol. To do this, specify H as the symbol substring starting position.
2. Add a string of zeros to the end of the reversed symbol. Add at least as many zeros as the number of characters in the field. For example, if the number field is six characters, add at least six zeros.
3. Truncate the result of step 2 to the required length (six characters in the example).
4. Reverse the result of step 3.

The following script shows how to do this to add leading zeros to the number in #NUMBER so that it fills a six-character field:

```
GET #TEMP = #NUMBER:H "000000"
GET #TEMP = #TEMP:6
GET #NUMBER = #TEMP:H
```

Right-Justifying the Contents of a Symbol

To ensure that the contents of a symbol are justified to the right of a field or a template, use one of the methods described here.

Method 1

Use a method similar to method 2 to add leading spaces to a symbol, so that when you load a field or a template with the value of the symbol, the value is justified to the right of the field.

For example, assume you have a field that is 10 characters long and you want to load it with the contents of a symbol #TEXT, so that the output is justified to the right of the field. The following script adds an appropriate number of spaces to the left of the text:

```
GET #TEMP = #TEXT:H "          "
GET #TEMP = #TEMP:10
GET #TEXT = #TEMP:H
```

You can then use the WRITE function to enter the value of #TEXT in the field of the data set.

Method 2

An alternative method of right-justifying the contents of a symbol is to specify the field length as the length, and Z as the starting position of the symbol substring.

For example, assume you have a template in which you want to display a column of three figures whose values are held in symbols #A, #B, and #C. If the maximum value of each of these figures is 999999, specify a substring length of 6, as in the following example template:

```
Line 1: <#A:6:Z>  
Line 2: <#B:6:Z>  
Line 3: <#C:6:Z>
```

If #A has a value of 35199, #B a value of 23, and #C a value of 123785, the output of the template will be as follows:

```
Line 1: 35199  
Line 2: 23  
Line 3: 123785
```

Programming Hints: Date and Time Fields

This article provides information on how to program date and time fields within ALL-IN-1. In particular, it describes how to do the following:

- Create date and time fields
- Validate date and time fields
- Convert dates between various formats
- Compare date fields

Creating Date Fields

ALL-IN-1 allows you to enter dates in several different formats. Users specify their preferred date format using the Set Working Conditions form; this information is held in the user profile. Wherever possible, ALL-IN-1 displays dates in an alphanumeric format, such as 15-Aug-1991 or Aug-15-1991. To create a date field for an alphanumeric date, use the FMS editor to create an 11-character field.

NOTE

ALL-IN-1 provides its own date validation facilities as described in the next section. Therefore, do not use the facility provided by FMS to create predefined date fields, since ALL-IN-1 does not process these fields correctly.

Validating Date Fields

ALL-IN-1 provides several special symbols and data set access blocks (DSABs) that you can use to validate dates. These symbols and DSABs are described in *Application Programming: Reference Volume 1*. The following named data shows how to use special symbols and validation DSABs to validate a date field, called BIRTH_DATE:

```
;;BIRTH_DATE;;  
/VALID=CAL$SET_DATE①:"BIRTH_DATE"②
```

- ^① CAL\$SET_DATE is a validation DSAB that ensures a valid date was entered; it does not set the current calendar date.

The CAL\$SET_DATE DSAB allows you to enter dates in several different formats, depending on the preferred date format specified in the user profile. These formats include the following:

- DD/MMM/YYYY — 15-May-1991
- DD/MM/YY — 15/05/91
- MM/DD/YY — 05/15/91
- A keyword, such as TODAY, TOMORROW, and TODAY+1W

CAL\$SET_DATE also lets you compute dates by units of days, weeks, or months. The following example shows how to compute the date of 7 days ago and put the result into a symbol #RESULT:

```
GET CAL$SET_DATE:"#RESULT".%WHOLE["TODAY -7D"]
```

APPLICATION PROGRAMMING

- ② This updates the field BIRTH_DATE with the validated date in the format specified in the user's working conditions.

Converting Date Formats

ALL-IN-1 allows you to convert dates from one format to another, by using the DATE_CONVERT function. DATE_CONVERT has the following syntax:

```
DATE_CONVERT dest-sym, source-sym, format key
```

For example, to convert the value of the current date (contained in the special symbol OA\$DATE) to month format and store the output value in the symbol #DATE, enter the following:

```
<DATE_CONVERT #DATE, OA$DATE, 6
```

If the value of OA\$DATE is 24/08/91, the value of #DATE is August.

You can also use keywords with DATE_CONVERT. The following example shows how to compute the date of 7 days ago in National Bureau of Standards (NBS) format:

```
<DATE_CONVERT #RESULT, "TODAY -7D", 7
```

For a list of the valid parameters for DATE_CONVERT, refer to *Application Programming: Reference Volume 2*.

Comparing Dates

It is often useful to be able to compare dates. For example, in an employee registration application that stores the name (NAME), employee number (NUMBER), and joining date (START_DATE) of all employees, you may want to compare certain dates so you can create an index of all the people who joined the company between the specified search dates.

The obvious way to create an index of records is to do a direct comparison between the contents of the field. With date fields, however, a direct comparison does not produce chronologically accurate results. This inaccuracy is caused by the way ALL-IN-1 processes dates. When ALL-IN-1 compares dates, it does a string comparison, which means that it compares the dates character by character, starting from the left.

As an example, assume that the application contains the following records:

Number	Name	Start Date
113457	John Smith	21-Jan-90
113514	Maria Braun	15-Jun-90
113645	Bill Harvey	10-Apr-91

A direct comparison of the contents of START_DATE would be as follows:

```
;;BIND;;  
  
BIND *STAFF TO STAFF_ENTRY WITH  
.START_DATE GES #SAVED_FROM_DATE AND  
.START_DATE LES #SAVED_TO_DATE
```

If the symbols #SAVED_FROM_DATE and #SAVED_TO_DATE contain the dates 10-Jan-90 and 20-May-90, respectively, ALL-IN-1 returns only the record whose joining date is 15-Jun-90. Clearly, this is chronologically inaccurate, but as a string value it falls between the specified dates.

To compare dates chronologically, you must first convert the dates into NBS format, using the DATE_CONVERT function.

The NBS format is a purely numeric representation of a date, starting with the year, then the month, day, hour, minute, second, and hundredth of a second. For example, the date 30-Jan-1991 13:55 is 1991013013550000 in NBS format.

By comparing dates in NBS format, ALL-IN-1 produces chronologically accurate data.

Therefore, to produce an index of records from a comparison of dates (using the staff registration application as an example), do the following:

1. Create a hidden field on the entry form to store the NBS format of the date entered in START_DATE.

To create a hidden field, use the FMS editor to create a 16-character field on the entry form. Give this field the FMS attributes No Echo and Display Only (so the user cannot modify the contents of the field), and assign a name to it, for example START_DATE_NBS.

2. Use DATE_CONVERT to convert the date from the START_DATE field to NBS format and store it in START_DATE_NBS.

Perform the date conversion when postprocessing the entry form. To do this, enter the following named data for the entry form:

```
;;.TYPE;;

ENTRY/MODE=UPDATE/HARD = .....
/POST=' IFEXIT\①.IF START_DATE NES "" THEN
  DATE_CONVERT START_DATE_NBS,START_DATE,7 ELSE
  GET START_DATE_NBS = "0000000000000000"②

;;START_DATE;;

/VALID=CAL$SET_DATE;GET START_DATE=OA$TM_DATE
/OPTIONAL
```

- ① Perform the date conversion when postprocessing the form. You can use the DATE_CONVERT function as a post function to the /VALID field qualifier, but this has the following disadvantages:

- If a user exits the field using BACKSPACE or UP ARROW, ALL-IN-1 does not perform the post function and, thus, does not update the NBS field.
- If a user changes an existing date (by using ERASE LINE to clear the field and then entering a new date), ALL-IN-1 does not update the value already loaded into the field containing the NBS format.

- ② An empty NBS field has a string value of " ". This value does not fall within the range "0000000000000000" to "9999999999999999." Therefore, you must specify that a blank NBS date field has the value "0000000000000000", otherwise the related record does not appear in the index search.

3. Create an index form that compares dates stored in NBS format.

APPLICATION PROGRAMMING

To compare dates from the START_DATE_NBS field, include named data like the following:

```
;;BIND;;  
XOP "~~SET_UP_NBS_DATES~~"  
BIND *STAFF_ENTRY TO STAFF_ENTRY WITH①  
.START_DATE_NBS GES #FROM_NBS AND  
.START_DATE_NBS LES #TO_NBS  
;;~~SET_UP_NBS_DATES~~;;②  
.IF #SAVED_FROM_DATE NES "" THEN  
DATE_CONVERT #FROM_NBS,#SAVED_FROM_DATE,7  
ELSE GET #FROM_NBS = "0000000000000000"\  
.IF #SAVED_TO_DATE NES "" THEN  
DATE_CONVERT #TO_NBS,#SAVED_TO_DATE,7  
ELSE GET #TO_NBS = "9999999999999999"
```

- ① The BIND record selection expression (RSE) uses the dates stored in the NBS date field to create the phantom data set. Prior to executing the BIND function, set up symbols to contain the NBS equivalents of the two search dates.
- ② The XOP ~~SET_UP_NBS_DATES~~ converts the dates to NBS format.

Creating Time Fields

ALL-IN-1 allows you to enter times in several different formats, including keywords such as NOON and MIDNIGHT. The user profile specifies whether times are displayed in 12- or 24-hour clock format. To create a time field for ALL-IN-1, use the FMS editor to create a seven-character field, and assign it a name, for example, MEAL_TIME.

NOTE

ALL-IN-1 provides its own time validation facilities, as described in the next section. Therefore, do not use the facility provided by FMS to create predefined time fields, since ALL-IN-1 does not process these correctly.

Validating Time Fields

ALL-IN-1 provides several special symbols and DSABs that you can use to validate times. These symbols and DSABs are described in *Application Programming: Reference Volume 1*. The following named data shows how to use special symbols and validation DSABs to validate a field called START_TIME:

```
;;START_TIME;;  
/VALID=CAL$TIME_CHECK①:"START_TIME"②
```

- ① CAL\$TIME_CHECK is a validation DSAB that validates a time. ALL-IN-1 checks the time against the user profile work start and end hours. So, if users work from 8:00 a.m. to 5:00 p.m., they can enter 3 to mean 3:00 p.m.
- ② This symbol contains the validated time in the format specified in the user profile.

Validating Start and End Times

If you have two time fields on a form, for example to schedule the start and end times of a meeting, you might want to validate that the end time is later than the start time. To validate time fields in this way, add the following named data to the form:

```
;;MEETING_TIME_BEGIN;;
/VALID=CAL$TIME_CHECK:"MEETING_TIME_BEGIN"
;;MEETING_TIME_END;;
/VALID=CAL$END_TIME_CHECK❶:"MEETING_TIME_BEGIN,MEETING_TIME_END"❷
```

- ❶ CAL\$END_TIME_CHECK validates an end time against a start time.
- ❷ The key of reference consists of two symbols, the first to contain the start time and the second to contain the validated time in the format specified in the user profile.

Programming an Application for Performance

This article gives guidelines on how to design and write applications that have the best performance, that is, that run as fast as possible and do not slow down the system.

The most important factors that affect the performance of an application are:

- Access to data files within ALL-IN-1
- Access to applications outside ALL-IN-1

When creating an ALL-IN-1 application, keep in mind that good performance often follows good design. An application that makes users move back and forth between forms and requires repeated screen paintings and other nonessential activities, not only frustrates users, but worsens performance. Conversely, the application that allows users to accomplish a task with as few steps as possible is generally the one with the better performance.

Real Versus Perceived Performance

Ensure that the design of your application meets the users' expectations. Often, perceived performance is more important than real performance: an application that performs well can appear slow to users, while an application that performs badly can appear fast.

For example, even though "Working..." messages add to the processing time, they may give the users the impression that the operations in progress are happening quickly.

Similarly, always collect data from the user before beginning processing. If you begin processing before asking for data, users wonder what is happening. This can mean that you must save data that could be operated on immediately, but users expect to see a fast response when they first call an application. Once they enter data, they are more prepared to wait a short time.

Use index forms where appropriate to your application. Generally, users are more productive when working from index forms. Although there may be some overhead in first displaying the index form, access to the records thereafter is much quicker.

Using FMS Forms

ALL-IN-1 is heavily form-driven. Even in hardcopy mode, all actions are based on the named data of the current form (or related forms, as in the case of form DEFAULT). Hard copy simply avoids repainting of screens. Therefore, fast access to forms is important for performance. To obtain the maximum performance from forms:

- Use the form libraries supplied by ALL-IN-1.

There is always overhead in opening form libraries. Do not invent new form libraries for new applications, unless they are necessary for controlling access to forms. Using the form libraries supplied by ALL-IN-1 for customizations reduces the overhead of opening form libraries.

- Where possible, keep form sizes below 4000 bytes.

When a form is accessed, FMS allocates workspace. The default workspace size is 4000 bytes. If a form is above this size, more workspace is allocated, but in segments, usually of 512-byte chunks, thus making the process of allocation longer.

APPLICATION PROGRAMMING

To see the size of the forms in a form library, use the following DCL command:

```
$ FMS/DIR/FULL formlibrary
```

Workspace is not allocated unless the form is displayed. Forms that are accessed for their named data only do not use workspace. Examples of this sort of form are **DEFAULT** and **EM\$INDEX\$OPTIONS**. If you have a form that is larger than 4000 bytes, consider putting some of its named data on another form that is not displayed. This is especially useful if you are writing an application in which the same options can be accessed from several forms.

- Try to keep the most commonly used options on the form that is displayed.

The way that **ALL-IN-1** searches for an option is as follows:

- When a form is compiled, the named data is arranged in alphabetical order of options. When the user inputs an option, the form is searched for that option.
- When a form references another form using the **/MORE** form qualifier, **ALL-IN-1** searches the first form until either:

- It finds the required option.
- It reaches an option that occurs later in the alphabet than the required option.

In this case, **ALL-IN-1** stops searching the first form and proceeds to search the next form until it finds the required option or reaches an option that occurs later in the alphabet than the required option.

ALL-IN-1 continues to search through each form until the required option is found.

Although this search does not use much central processing unit (CPU) resource, it may require named data to be paged in while forms are searched. This increases the page fault rate.

When a form is compiled, hidden menu options executed by the **XOP** function are placed at the end of the form and are found last in a search.

- Always call forms explicitly with the **FORM** function.

Although a user can access a form directly by entering its name in the choice field of a menu, performance is better if you provide an option in named data that calls the form using the **FORM** function.

For example, to access form **XYZ**, the user can enter **XYZ** in the choice field of any menu. However, if the current menu contains an option **XYZ**, which simply performs the function **FORM XYZ**, performance is improved. This is because of the way that **ALL-IN-1** processes user input in the choice field of a menu, as described in the chapter on menus in the *Application Programming: Guide*.

If an application requires users to move frequently from a menu (for example, **A**) to another form (for example, **B**), include an entry **B** with the form function to call form **B** in the named data of menu **A**. The option calling **B** need not appear on the menu.

- Limit the number of forms that your application displays.

After a form is displayed, it is held in cache. Until the cache is full, the workspace for a form is held in memory, even when the form is not being used. Thus, if you call a form unnecessarily, the workspace for that form occupies memory until the cache is full.

The number of forms held in cache is determined by a link time constant, called `OA$DSA_FORM_CACHE_SIZE`, set in `A1LINK.COM`. Although it may be changed by editing `A1LINK.COM` and relinking the `ALL-IN-1` image, the optimum value for normal work was determined to be 12 forms.

Designing Forms

When designing forms, consider the following guidelines:

- Consider the overhead cost of special screen formatting features.

FMS gives the application designer a variety of special features from which to choose, such as bolding, blinking, reverse video, and drawing boxes.

These features, however, have a cost in terms of the overhead required to do the screen painting. If quick screen painting is desired, avoid these special FMS features.

- Avoid a large number of fields in a single form or form set. `ALL-IN-1` loads forms field-by-field, not all at once.
- Let users perform an operation without calling a form.

Often, a single operation requires certain information before it can proceed. For example, when selecting a document, the `SEL` option requires at least a folder name. Normally, you obtain this information by displaying an argument form. However, when users need to enter only a single item, it improves performance if they can enter that item without the overhead of calling another form. In the case of the `SEL` option, this is achieved by using the `OA$MENU_REMAINDER` symbol to let the user enter `SEL folder name` in the choice field of a menu. See the named data of the `EM` menu for an example of how to do this.

Writing Efficient Scripts

When you write a script, bear in mind the following guidelines:

- Use the `ALL-IN-1` functions and qualifiers throughout, rather than reverting to `DIGITAL` Command Language (DCL) or user-written application code. In this way, you avoid leaving the current `ALL-IN-1` process or the context of the current field or form.
- Use the following guidelines for `.GOTO` and `.LABEL` script directives.

The first time the script processor meets a `.LABEL` directive, it creates an in-memory table to hold the addresses of labels. It stores any labels it finds as it processes each line. When it comes to a `.GOTO`, it searches the in-memory table and then goes forward through each line of the script until it finds the label. It stores any other labels it finds on the way. However, the in-memory table can hold only 32 entries. Therefore:

- Always start a script with a label to ensure that the in-memory table is created immediately.
- Never have more than 32 labels in a script.
- Move backwards with `.GOTO` when possible. A backward `.GOTO` processes faster than a forward `.GOTO`. The further a forward `.GOTO` is from its `.LABEL`, the worse it is.
- Design the script so that the most common path is the one without branching.

APPLICATION PROGRAMMING

- Do not use the expression `GET OA$FUNCTION = "function"`, where the function will work by itself. `GET OA$FUNCTION` is only necessary to supply a symbol to a function that demands a value. Similarly, avoid the `.FUNCTION`, `.FX`, and `.FZ` directives in a `DO` script, where they are unnecessary, because they cause `GET` to be performed unnecessarily.
- Do not fill a form field from a script. Although this is possible, it is much more expensive than filling the field from within the form itself.
- An alternative to a script is the use of the `XOP` function to include the script commands within the named data of a form. Use this alternative when the script is simple and small. In general, processing of functions is faster from named data, but be careful not to make forms that are frequently displayed too large.

The `XOP` function is also useful for pieces of code that are common to several options.

- When checking the value of `OA$STATUS`, use `IFSTATUS` or `IFNOTSTATUS` in preference to the `.IF OA$STATUS EQUAL ...` directive. This also applies to named data.
- Make sure `FOR` functions stop as soon as possible. You can use `IFSTATUS` to stop a `FOR` loop as follows:

```
GET OA$STATUS = 1
FOR A DO .IF .x = #x THEN GET OA$STATUS = 0 \IFSTATUS
```

As soon as `.x` begins with `#x`, the loop stops.

- Use the Customization Management (CM) site text library, `CMTXL.TXL`

The `DO` scripts, `SCRIPT` scripts, and `MERGE` templates used by your application can be stored in compiled format in the site text library. Accessing the compiled version in the site text library is much faster than accessing the script or template directly. Files for inclusion in the CM site text library are kept in the `OA$SITE_DO_*`, `OA$SITE_SCP_*`, and `OA$SITE_BLP_*` directories. It is the `ALL-IN-1` manager's job to ensure that the files in these directories are compiled into the site text library.

Once a file is compiled into the site text library, you call the compiled version by specifying only the name of the file. If you specify a directory containing the file, `ALL-IN-1` uses the version in that directory and does not look for a version in the site text library.

If it is not appropriate to include a file in the site text library, for example, because it is used by only a small number of users, you may want to modify the symbols `OA$TXL_SEARCH_ORDER` and `OA$FILE_SEARCH_ORDER`. When you call the `DO`, `SCRIPT`, or `MERGE` function, `ALL-IN-1` looks for the file in directories specified by these two symbols. These symbols are defined for users within their own individual `ALL-IN-1` process, but may be defined dynamically within any application. Modifying the search order for files, by modifying these symbols on a user-by-user basis or application-by-application basis, can result in a significant reduction in file lookups and an improvement in performance. Refer to *ALL-IN-1 Application Programming: Reference Volume 1* for a full description of these symbols.

If your script will not be included in the site text library, do not use long comments within it. You can, however, put comments at the end of the script without any performance penalty.

- In `ALL-IN-1` Version 3.0, two new functions are introduced: `<INCREMENT` and `<DECREMENT`. Use these, rather than the `<COMPUTE` function, when incrementing or decrementing the value of a symbol.

Using Symbols

When writing scripts, named data, command procedures, and other application software, symbolic data is frequently used. ALL-IN-1 provides a number of ways of referencing this data:

- String literals — “Dog”
- Local symbols that are memory-resident — #DOG
- Permanent symbols that are stored in a Record Management Services (RMS) file — \$DOG
- Field names that reference data on the current form — DOG
- Special symbols that are maintained by ALL-IN-1 and may be readable, writable, or both
- Data Set References (DSRs) that point to fields in records of a data set — PROFIL.DIRECT[OA\$USER]
- Symbol substitution — @#CANINE = #DOG = “Dog”

Although all symbol types can be used interchangeably where symbolic data is acceptable to ALL-IN-1, it is often possible that the desired information resides in more than one place (or can be stored in more than one place), and that one place is faster to retrieve from than the other.

- Temporary and permanent symbols compared

Because temporary symbols are memory-resident and permanent symbols are stored in an RMS file, it is more efficient to write scripts, named data, or user-defined processes (UDPs) that reference temporary symbols rather than permanent symbols. Of course, if you want to maintain the value of a symbol from one session to the next, you must store that value in a permanent symbol, since temporary symbols are not maintained from session to session. However, even in this case, it is faster to retrieve the permanent symbol at the beginning of a script, store it in a temporary symbol for the duration of the procedures, and finally store the result in the permanent symbol table.

If too many permanent symbols are created, access time to the permanent symbol table file (.PST) will become slower, unless the block size of the file is increased. A block size of 6 is recommended.

- Special symbols

Many of the ALL-IN-1 special symbols are local representations of data in several of the key ALL-IN-1 data files, such as the user profile or the user's File Cabinet. Rather than use a full data set reference to retrieve the information, you can reference the special symbol, which improves performance. An example is to use OA\$PROFIL_DIRECT, rather than PROFIL.DIRECT[OA\$USER].

Using the Subprocess

When ALL-IN-1 executes a DCL command or command procedure, a subprocess is created. This has a large impact on performance (for example, because of the buffered input/output (I/O) caused by mailbox communications). Therefore, where possible, avoid using the subprocess.

APPLICATION PROGRAMMING

In normal processing, ALL-IN-1 does not use the subprocess. Once a subprocess is created, it remains for the remainder of the session. To avoid using a subprocess:

- Use ALL-IN-1 functions, rather than the equivalent DCL commands, for example, EDIT, PURGE_FILE, RENAME, DELETE_FILE, COPY, QUEUE_BATCH.
- Use the ALL-IN-1 INSTALL and EXECUTE functions to create your own functions. By doing so, you avoid running images from the subprocess.
- If you do not want the subprocess and the main ALL-IN-1 process to communicate, use SPAWN to create a detached process. You can then use the ATTACH command and the SPAWN command to toggle between your main and detached processes.

If you must use the subprocess, use it as little as possible.

Testing Application Performance

The following are suggestions on how to measure the resources used by your application.

- Use the Element Speed Measurement (ESM) option on the CM menu to test keystrokes.
- Use the DECLARE_METER function to place metering hooks in the code and, if you do not consider what you are metering to be a major function, remove the hooks before you move the application to the live area. See the *ALL-IN-1 Application Programming: Reference, Volume 2* for details on the DECLARE_METER function. Also see the *ALL-IN-1 Management Guide* for a description of the metering facility.
- Use the OA\$STAT_GET_STATISTICS function to get a snapshot of resource usage.

Among the symbols set by this function the following will be useful:

STAT\$CPUTIM — CPU time
STAT\$DIRIO — direct I/Os
STAT\$PAGFLTS — page faults
STAT\$BUFIO — buffered I/Os

- Calculate elapsed time using the OA\$DATE_NBS symbol.
To find the time taken for any part of your application to run, get the value of the symbol OA\$DATE_NBS immediately before and immediately after your code. The elapsed time is the difference between these two values.
- Use the ALL-IN-1 OA\$LOG_TO_FILE function to give you the elapsed time and resource usage between keystrokes and functions.
- Use the Trace facility. For example, by using OA\$TRA_SET IO, you can see whether a record is fetched directly by key or after a sequential search.

Using Metering

ALL-IN-1 provides the means to collect statistics on resource use of all the major activities by means of metering. The metering collection points also act as DECtrace collection points for customers with DECtrace. The information collected by the meters can be used as a basis for performance analysis and capacity planning.

For consistency, include metering collection points in your code. It is advisable to meter only major functions. For example, use metering for actions such as reading a document or sending a message, but not for validating an address. If you use too many metering collection points, performance of the application suffers, and the ALL-IN-1 manager becomes swamped with information.

Programming for Optimal Field Access

This article describes different ways of accessing data that is held in an ALL-IN-1 data set. It also gives hints on how to achieve the fastest data access in your application.

Searching for Records

There are two ways in which ALL-IN-1 searches a data set to find a record: using a sequential search or using an indexed search.

Using a sequential search, ALL-IN-1 examines every record in the data set to find the desired record. With a large data set, a sequential search can be slow.

Using an indexed search, ALL-IN-1 makes use of the key structure of RMS files. An RMS file contains an index of the values held in the key field of each record. This index is known as the key of reference, and the entry for an individual record in the key of reference is known as the key to that record. When you access a data set, you specify the key of reference to use, and the key value of the record you want to access. ALL-IN-1 searches the key of reference until it finds a key that matches the key you supply. It then accesses the record that corresponds to that key, instead of examining every record. This is known as an indexed search and is usually much faster than a sequential search.

Accessing Data

There are several ways of accessing fields from records in a data set. These are:

- Using a DSR
Use a DSR to access a single field from an individual record.
- Using the %OPEN and %CLOSE special fields in conjunction with a DSR
Use these special fields to access several fields in the same record.
- Using a record selection expression (RSE) with the FOR function
Use an RSE in a FOR loop to select multiple fields from contiguous records.
- Using DATA_FILE subfunctions
Use these subfunctions to access multiple fields from multiple records.

Always use the simplest method that is available for the type of access you require. For example, there is no point in using DATA_FILE subfunctions to access fields in a single record.

Using a DSR

The simplest way to read data from a file is with a DSR. A DSR has the following syntax:

```
dsab.kor.field[key]
```

where:

dsab is the name of the DSAB that accesses the data set.

APPLICATION PROGRAMMING

kor is the name of an RMS key of reference to the data set. This is optional when accessing by primary key.

field is the name of the field that you want to access.

key is an ALL-IN-1 symbol whose value is the RMS key of the record to be accessed.

In the following example, the DSR accesses the DESC field from the FORMAT data set for the record whose key value is WPSPLUS:

```
FORMAT.DESC["WPSPLUS"]
```

Using %OPEN and %CLOSE

If you want to access more than one field from a single record in a data set, use the special fields %OPEN and %CLOSE within your DSR. For example, to access the DESC, DEFAULT_FILE, and DSAB fields from the FORMAT data set for the record whose key value is WPSPLUS:

```
GET FORMAT.%OPEN["WPSPLUS"]
GET .DESC
GET .DEFAULT_FILE
GET .DSAB
GET .%CLOSE
```

Using an RSE

Use RSEs to select multiple fields from contiguous records in a data set. For example, to access the DESC, DEFAULT_FILE, and DSAB fields from the records whose key values begin with ASCII, use an RSE with the FOR function as follows:

```
FOR FORMAT WITH .%KEY BEGINNING "ASCII" DO GET .DESC "," .DEFAULT_FILE "," .DSAB
```

To allow ALL-IN-1 to perform an indexed search, the RSE must specify the key value, or at least the beginning of the key value, and the name of the FMS field that contains the key value. If you do not specify these in your RSE, ALL-IN-1 cannot access the records by key and, therefore, looks through the data set sequentially until it finds a record that matches the search criteria. This makes accessing the data set much slower and results in poorer performance.

When you write an RSE, bear in mind the following rules:

- Use only the EQS, ENS, or BEGINNING relational operators. If you use any other relational operator, ALL-IN-1 performs a sequential search of the whole data set.
- Use the AND operator where possible, instead of the OR operator.
- If you specify an alternate key, specify the name exactly as it is quoted in the File Definition Language (FDL).
- Reference the FMS field that maps to the first segment of the key by name, rather than using the special field name %KEY.

It is not always possible to follow these rules, but if you do not follow them, you will degrade performance.

ALL-IN-1 sometimes performs a search that is partially indexed and partially sequential. For example, if your RSE uses EQS or BEGINNING when searching for the key field, ALL-IN-1 performs an indexed search. However, if the RSE also contains a Boolean expression that uses another relational operator (such as CONTAINING), ALL-IN-1 performs a sequential search of the records whose key fields match.

To check whether ALL-IN-1 is doing an indexed or a sequential search, use OA\$TRA_SET IO to turn on I/O tracing while you are testing an application. The resulting log shows the record accesses, and you can determine whether the application was coded in the optimum manner. Sequential searches show repeated GET_NEXT operations being performed while indexed searches result in LOCATE and GET_BY_KEY operations.

Using DATA_FILE Subfunctions

If you want to access fields from several records in a data set and the records are not contiguous, use DATA_FILE subfunctions. The following example accesses the DESC, DEFAULT_FILE, and DSAB fields from the FORMAT data set for the records whose key values are WPSPLUS and ASCII WPS:

```
DATA_FILE OPEN FMT FORMAT
DATA_FILE GET FMT "WPSPLUS" #WPREC
DATA_FILE GET_FIELD FMT DESC #WPDESC
DATA_FILE GET_FIELD FMT DEFAULT_FILE #WPDFT
DATA_FILE GET_FIELD FMT DSAB #WPDSAB
DATA_FILE GET FMT "ASCII WPS" #AWREC
DATA_FILE GET_FIELD FMT DESC #AWDESC
DATA_FILE GET_FIELD FMT DEFAULT_FILE #AWDFT
DATA_FILE GET_FIELD FMT DSAB #AWDSAB
DATA_FILE CLOSE FMT
```

Accessing Data for Field Validation

ALL-IN-1 also searches a data set when you specify validation on a field, using /VALID or /RSE_VALID. Therefore, try to ensure that the search is as efficient as possible.

Whenever possible, use the key field as part of the search. In a /VALID statement, use the key field within the DSAB.FIELD syntax. In a /RSE_VALID statement, specify the key field within the RSE. This applies also in the case of a segmented key, but only to the first segment of the key. When referencing this first segment, ALL-IN-1 does indexed searching; referencing any other segment results in a full sequential search of the data set.

The action performed by each validation qualifier is equivalent to a FOR function as follows:

- /VALID=DSAB.FIELD

equates to:

```
FOR DSAB WITH .FIELD == @OA$FLD_NAME DO OA$VAL_SET_VALID
```

When you use the /VALID qualifier, ALL-IN-1 generates an RSE. In this case, the RSE is DSAB WITH .FIELD == @OA\$FLD_NAME.

If FIELD is the key field, ALL-IN-1 does an indexed search. If it is not the key field, ALL-IN-1 does a sequential search.

APPLICATION PROGRAMMING

- **/RSE_VALID = DSAB.FIELD**

equates to:

```
FOR DSAB DO .IF .FIELD == @OA$FLD_NAME THEN OA$VAL_SET_VALID
```

When you use the **/RSE_VALID** qualifier, **ALL-IN-1** does not generate an RSE. In this example, no RSE was specified as a parameter. Since **ALL-IN-1** does not generate one, it does a sequential search of the whole DSAB. The field name **FIELD**, appended to the DSAB, is not used as a search criterion.

- **/RSE_VALID=DSAB.FIELD WITH .A == A and .B == B**

equates to:

```
FOR DSAB WITH .A == A AND .B == B DO IF .FIELD == @OA$FLD_NAME THEN OA$VAL_SET_VALID
```

In this example, **ALL-IN-1** will do an indexed search if **A** or **B** is the key field or the first segment of the key field. Field **FIELD** is not part of the RSE, so does not affect the type of search.

Working with Compound Documents in ALL-IN-1

This article describes the functions that were introduced in ALL-IN-1 Version 2.4 and ALL-IN-1 Version 3.0 to allow application programmers to work with compound documents.

Introduction to Compound Documents

A compound document is a file, or number of files, that may contain a number of integrated components including text, graphics, and scanned images. ALL-IN-1 includes support for compound documents that use the CDA architecture, which defines how one or more complex files can be structured as a single compound document.

In compound documents that consist of more than one file, the main file contains *external references* that point to other component files. These component files may, in turn, contain external references to other files. External references are also sometimes known as *live links*. Compound documents that consist of more than one file are known as *composite compound documents*. When a user works on such a document, the referenced files are presented as a single document.

When a composite compound document is moved, the component files may remain in their original location or they may be moved with the main document. Whether a component file is moved or not depends on an attribute of the external reference to that file. When an external reference has a COPY attribute, the component file is transferred with the main document. When an external reference has a NO COPY attribute, the component file is not moved with the main document. The COPY/NO COPY attribute is set by the application that creates the file.

Compound documents have the RMS attribute, *stored semantics*. The value of the stored semantics attribute is the file's *semantic tag*, which specifies how file data is to be interpreted.

ALL-IN-1 can currently handle the following compound document storage formats:

- Digital Document Interchange Format
DDIF is the standard CDA encoding format. DDIF documents can contain a combination of graphics and text. Documents created using DECwrite have this format.
- DTIF
DTIF is the standard CDA format for tables. Documents created using DECdecision or DECcalc have this format.
- Data Object Transport Syntax
DOTS is the format for packing and transferring composite compound documents as a single file.

DOTS format files must be unpacked before users can work with the documents. When a user uses a DOTS format file in ALL-IN-1, the component files are temporarily unpacked. DOTS format files are permanently unpacked when they are sent to VMS.

APPLICATION PROGRAMMING

Additional Information

For information about semantic tags, refer to the following documents:

- *Guide to VMS File Applications*
- *VMS Record Management Services Manual*

For more information about CDA, refer to the following documents:

- *Getting Started with the CDA Converter Library*

This document provides an introduction to the CDA Converter Library input and output formats that you can use with compound document applications.

- *Guide to the CDA Converter Library*

This gives more detailed information on the CDA Converter Library input and output formats.

Application Programming Interface to Compound Documents

To allow application programmers to work with compound documents, ALL-IN-1 includes the following functions:

- CDA_CLEANUP
- CDA_CONVERT
- CDA_PACK
- CDA_UNPACK
- GET_RMS_SEMANTIC_TAG

These functions are described on the following pages.

CDA_CLEANUP

Syntax

CDA_CLEANUP file-sym [,delete-flag-sym]

Description

The CDA_CLEANUP function frees the memory used by the CDA_UNPACK function and, optionally, deletes the unpacked files.

When the CDA_UNPACK function is used to unpack files into a VMS directory, the function allocates memory to hold the list of files created. To free this memory, call the CDA_CLEANUP function each time you use CDA_UNPACK in this way. You cannot use the CDA_CLEANUP function to free the memory used for more than one CDA_UNPACK call.

Parameters

file-sym

This parameter is an ALL-IN-1 symbol that specifies the name of the root CDA file. Use the value returned in the symbol specified in the *root-sym* parameter of the CDA_UNPACK function.

delete-flag-sym

This optional parameter is an ALL-IN-1 symbol that specifies whether the unpacked files are to be deleted. If *delete-flag-sym* has a value of 1, ALL-IN-1 deletes the unpacked files. If *delete-flag-sym* has any other value or is not specified, no files are deleted.

Examples

See the examples in the section on CDA_UNPACK.

Special Features and Restrictions

Do not call CDA_CLEANUP if the DOTS format file was unpacked into the ALL-IN-1 File Cabinet.

CDA_CONVERT

Syntax

CDA_CONVERT infile-sym, infmt-sym, outfile-sym, outfmt-sym
[,optfile-sym]

Description

The CDA_CONVERT function converts a revisable-format input file to create an output file with the specified storage format.

Parameters

infile-sym

This parameter is an ALL-IN-1 symbol specifying the name of the input file to be converted.

infmt-sym

This parameter is an ALL-IN-1 symbol that specifies the format of the file specified in *infile-sym*.

Specify any format for which your CDA Converter Library contains a front end. For more information on CDA converters, see the CDA documentation.

outfile-sym

This parameter is an ALL-IN-1 symbol specifying the name of the output file to be created. The default file type is .DDIF.

outfmt-sym

This parameter is an ALL-IN-1 symbol specifying the format of the output file.

Specify any format for which your CDA Converter Library contains a back end. For information on CDA converters, see the CDA documentation.

optfile-sym

This optional parameter is an ALL-IN-1 symbol that specifies the name of an options file containing the input and output processing options to be applied during the conversion. These processing options affect how the input file is processed and how the output file is created or displayed.

The options file has a default file type of .CDA\$OPTIONS.

For more information about options files, refer to the CDA Converter Library documentation.

CDA_CONVERT

Examples

The following example converts the file MYFILE.DDIF from DDIF format to a PostScript output file NEWFILE.PS in the user's default login directory:

```
CDA_CONVERT "MYFILE.DDIF", "DDIF", "SYS$LOGIN:NEWFILE.PS", "PS"
```

The following example uses the options file, DX.CDA\$OPTIONS, to convert the file specified in the symbol #INFILE from the format specified in the symbol #FMT, creating a DDIF output file TEMP.DDIF.

```
CDA_CONVERT #INFILE, #FMT, "TEMP", "DDIF", "DX.CDA$OPTIONS"
```

Special Features and Restrictions

You must have the CDA Converter Library for VMS installed on your system to use the CDA_CONVERT function. Each converter that you want to use must be installed as a known image.

CDA_PACK

Syntax

CDA_PACK *infile-sym*, *outfile-sym*, *tag-sym*

Description

The CDA_PACK function packs a composite compound DDIF or DTIF document, together with all copy-referenced files, into a DOTS format file.

Parameters

infile-sym

This is an ALL-IN-1 symbol containing the RMS file specification of the file to be packed.

outfile-sym

This is an ALL-IN-1 symbol containing the RMS file specification of the packed DOTS format file to be created. If *outfile-sym* does not include a device or directory specification, ALL-IN-1 creates the DOTS format file in the current directory.

tag-sym

This is an ALL-IN-1 symbol that receives the semantic tag name of the created file. If the file is successfully packed into a DOTS format file, *tag-sym* contains the value DOTS after the CDA_PACK call.

Examples

The following example packs the file specified in the symbol #DDIF_FILE into a DOTS format file OA\$TEMP:PACKED.DOTS. If the file is successfully packed, the symbol #NEWTAG receives the value DOTS.

```
CDA_PACK #DDIF_FILE, "OA$TEMP:PACKED.DOTS", #NEWTAG
```

Special Features and Restrictions

If the file specified in *infile-sym* has no copy-references, CDA_PACK copies the file to the one specified in *outfile-sym* and loads the symbol specified in *tag-sym* with the semantic tag name of the copied file.

CDA_UNPACK

Syntax

CDA_UNPACK infile-sym, dest-sym [,root-sym] [,list-flag-sym]
[,dest-type-sym]

Description

CDA_UNPACK unpacks a DOTS format file, placing copies of its constituent DDIF and DTIF files in a VMS directory or in a File Cabinet folder. Use this function if you are using a CDA application that cannot read DOTS format files.

Parameters

infile-sym

This is an ALL-IN-1 symbol that specifies the RMS file specification of the DOTS format file to be unpacked.

dest-sym

This is an ALL-IN-1 symbol that specifies where the unpacked files are to be placed. The *dest-sym* parameter can be either of the following:

- Name of a folder in the current File Cabinet drawer
- VMS directory

If you do not specify *dest-type-sym* or if *dest-type-sym* has a value of VMS, *dest-sym* must specify a VMS directory.

If *dest-type-sym* has a value of CAB, *dest-sym* must specify a folder name.

root-sym

This optional parameter is an ALL-IN-1 symbol to which the location of the root CDA file is written. If the unpacked files are placed in a VMS directory, *root-sym* receives the full expanded RMS file name of the root file. If the unpacked files are placed in a File Cabinet folder, *root-sym* receives the FOLDER/DOCNUM of the root file.

list-flag-sym

This optional parameter is an ALL-IN-1 symbol that specifies whether ALL-IN-1 generates a list of the unpacked files or documents in the message buffer. If *list-flag-sym* has a value of 1, ALL-IN-1 generates a list; if *list-flag-sym* has any other value or is not specified, ALL-IN-1 does not generate a list.

Depending on the value of *dest-type-sym*, the list contains file names or FOLDER/DOCNUM values.

dest-type-sym

This optional parameter is an ALL-IN-1 symbol that specifies where the unpacked files are to be placed. Possible values of *dest-type-sym* are:

- VMS — This specifies that the unpacked files are to be placed in a VMS directory
- CAB — This specifies that the unpacked files are to be placed in the ALL-IN-1 File Cabinet

If you do not specify *dest-type-sym*, ALL-IN-1 places the unpacked files in a VMS directory.

Examples

The following script temporarily unpacks the DOTS format file specified in symbol #DOTSFILE into the directory defined by the logical name OA\$TEMP. A list of the file names of the unpacked files is loaded into the message buffer. When the script PROCESS_CDA finishes using the unpacked files, CDA_CLEANUP frees the memory used by the unpack operation and deletes the unpacked files.

```
CDA_UNPACK #DOTSFILE, "OA$TEMP:", #ROOTFILE, 1
.IF OA$STATUS NE 1 THEN .EXIT
DO PROCESS_CDA
CDA_CLEANUP #ROOTFILE, 1
.EXIT
```

In the next script extract, CDA_UNPACK permanently unpacks the DOTS format file and places the unpacked files in the VMS directory specified in the symbol #UNPACK_DIR:

```
CDA_UNPACK #DOTSFILE, #UNPACK_DIR, #ROOTFILE, 1
.IF OA$STATUS NE 1 THEN .GOTO ERR_IN_UNPACK
CDA_CLEANUP #ROOTFILE, 0
```

The following example unpacks a DOTS format file and places the unpacked files in the File Cabinet folder CDA DOCUMENTS:

```
CDA_UNPACK #DOTSFILE, "CDA DOCUMENTS", #ROOTDOC, 1, "CAB"
```

Special Features and Restrictions

If the file specified in *infile-sym* is not in DOTS format, CDA_UNPACK returns the full file specification of this file in the symbol *root-sym*.

If the unpacked files are placed in a VMS directory, CDA_UNPACK maintains in memory a list of files created by this function call. To free this memory, call the CDA_CLEANUP function, using the value returned in *root-sym* as the *file-sym* parameter.

Not all versions of CDA support the File Cabinet as the unpack destination.

GET_RMS_SEMANTIC_TAG

Syntax

GET_RMS_SEMANTIC_TAG [/VALUE] file-sym, tag-sym

Description

This function returns the semantic tag of a file. The function can return the name of the tag (for example, DDIF) or its numeric value, in the form of a hexadecimal number. The following table shows the numeric forms of the DDIF, DOTS, and DTIF tags.

Tag Name	Number
DDIF	2B0C8773010301
DOTS	2B0C8773010302
DTIF	2B0C8773010303

Parameters

file-sym

This parameter is an ALL-IN-1 symbol specifying the file whose semantic tag is to be returned.

If the file specified as *file-sym* has no semantic tag, ALL-IN-1 returns a null string.

tag-sym

This parameter is an ALL-IN-1 symbol to which the tag name or numeric value is to be written.

Qualifier

/VALUE

If you specify this optional qualifier, ALL-IN-1 returns the numeric form of the semantic tag, rather than the name of the tag.

GET_RMS_SEMANTIC_TAG

Examples

The following example returns the semantic tag of FILE1.DDIF in the user's default login directory and enters the name of the tag in the field named FIELD3 of the current form.

```
GET_RMS_SEMANTIC_TAG "SYS$LOGIN:FILE1.DDIF", FIELD3
```

The next example returns the numeric form of the semantic tag of the file specified in the symbol #WORK_FILE and displays this value on line 24 of the screen.

```
GET_RMS_SEMANTIC_TAG /VALUE #WORK_FILE, OA$DISPLAY
```

Special Features and Restrictions

GET_RMS_SEMANTIC_TAG sets OA\$STATUS to 1 if it successfully opens the file specified in *file-sym*. Otherwise, it sets OA\$STATUS to 0.

If the file specified in *file-sym* has a semantic tag that is unknown to your VMS system and you do not use the /VALUE qualifier, GET_RMS_SEMANTIC_TAG returns a null string.

APPLICATION PROGRAMMING

Using the .FUNCTION and .FX Script Directives in ALL-IN-1 Scripts

This article describes the difference between the .FUNCTION and .FX script directives. It also contains several example scripts that illustrate how ALL-IN-1 processes .FUNCTION and .FX.

ALL-IN-1 Scripts

ALL-IN-1 has DO mode scripts and SCRIPT mode scripts. Their main characteristics are:

- DO scripts

DO mode scripts can contain ALL-IN-1 functions and script directives. You use a DO script to execute procedures with ALL-IN-1 applications. In effect, they perform the same tasks as DCL command procedures.

ALL-IN-1 processes DO scripts immediately.
- SCRIPT scripts

SCRIPT mode scripts store user input, for example, keynames, options, and text. You use a SCRIPT script to provide ALL-IN-1 with input that a user would normally enter interactively, for example, entering an option at a menu. UDPs are examples of SCRIPT scripts.

ALL-IN-1 does not process SCRIPT scripts immediately. Whenever you invoke a SCRIPT script, ALL-IN-1 places it on a SCRIPT script stack. A SCRIPT script remains unprocessed until ALL-IN-1 requires user input.

For more information on script stacks, refer to the chapter on scripts in the *ALL-IN-1 Application Programming: Guide*.

Using the .FUNCTION and .FX Script Directives

Note that .FUNCTION and .FX are redundant in DO scripts. Both .FUNCTION and .FX enable you to call an ALL-IN-1 function or list of functions for use in a SCRIPT mode script. For example, you can have the following line in a SCRIPT script:

```
.FUNCTION DISPLAY This an extract from a SCRIPT script\FORCE
```

In this example, .FUNCTION calls the ALL-IN-1 functions DISPLAY and FORCE.

Although you can use both .FUNCTION and .FX to call ALL-IN-1 functions, they are not identical. The difference between .FUNCTION and .FX is most important if you use them to call an ALL-IN-1 function that requires user input.

If you use .FUNCTION to call a function that requires user input, ALL-IN-1 takes the input from the current SCRIPT script. If you use .FX to call a function that requires user input, however, ALL-IN-1 expects the user to supply the input.

To illustrate the different behavior of .FUNCTION and .FX, create two nearly identical SCRIPT scripts, both of which call a function that requires user input; the only difference is that one uses the .FUNCTION directive to call the function, and the other uses .FX.

APPLICATION PROGRAMMING

FU_PROMPT.SCP is an example SCRIPT script that uses the .FUNCTION directive.

```
! This SCRIPT script is called FU_PROMPT.SCP
.LABEL START
WP
{CR}
.FUNCTION YESNO_PROMPT " Do you want to continue? [Y/N] "
Y
{CR}
.EXIT
```

If you run FU_PROMPT.SCP using the SCRIPT function, ALL-IN-1 does not wait for you to supply a response to the prompt. ALL-IN-1 takes the user input it requires from the script itself.

FX_PROMPT.SCP is an example of a SCRIPT script that uses the .FX directive.

```
! This SCRIPT script is called FX_PROMPT.SCP
.LABEL START
WP
{CR}
.FX YESNO_PROMPT " Do you want to continue? [Y/N] "
Y
{CR}
.EXIT
```

If you run FX_PROMPT.SCP using the SCRIPT function, ALL-IN-1 does not take the user input it requires from the script. Instead, ALL-IN-1 waits for you to supply a response to the prompt.

When you run FX_PROMPT, ALL-IN-1 does the following:

1. Displays the Word and Document Processing menu.
2. Displays a prompt asking if you want to continue.
3. Suspends script processing, places the SCRIPT script on the script stack, and returns control to you to supply the input interactively.
4. Resumes script processing when you enter a response, and supplies the rest of the script (Y and a carriage return) as user input to ALL-IN-1. Since Y is not a valid menu option, this causes ALL-IN-1 to display the following error message:

```
Invalid choice - re-enter
```

In effect, the .FX directive hides that there is an active SCRIPT script from the function it calls.

Invoking Other Scripts Using .FUNCTION and .FX

You can use .FUNCTION and .FX to call any ALL-IN-1 function. Thus, if you use .FUNCTION or .FX to call the DO or the SCRIPT functions, you can call another script from within a SCRIPT script.

If you use .FUNCTION or .FX to call a SCRIPT or a DO function, which in turn calls a script containing a function that requires user input, you should note the difference between using .FUNCTION and .FX.

To illustrate how ALL-IN-1 processes scripts called by .FUNCTION or .FX, create the following SCRIPT script (FU_EXAMPLE.SCP), which calls a DO script (INPUT_WANTED.SCP), using the .FUNCTION directive:

```
! This SCRIPT script is called FU_EXAMPLE.SCP

.LABEL START
.FUNCTION DO INPUT_WANTED
Y
{CR}
.EXIT
```

The DO script INPUT_WANTED contains the following:

```
!This DO script is called INPUT_WANTED.SCP

.LABEL START
YESNO_PROMPT
.EXIT
```

When you invoke FU_EXAMPLE, using the SCRIPT function, ALL-IN-1 does the following:

1. Executes INPUT_WANTED with a DO function call. This means that any functions or directives in INPUT_WANTED execute in DO mode.
2. Displays a prompt asking if you wish to continue. Since you called the script containing the YESNO_PROMPT function using .FUNCTION, ALL-IN-1 takes the input it requires for the prompt from the SCRIPT script FU_EXAMPLE.SCP.

To see how .FUNCTION differs from .FX when calling other scripts, create the following SCRIPT script(FX_EXAMPLE.SCP), which uses .FX to call the DO function:

```
! This SCRIPT script is called FX_EXAMPLE.SCP

.LABEL START
.FX DO INPUT_WANTED
Y
{CR}
.EXIT
```

When you invoke FX_EXAMPLE.SCP, using the SCRIPT function, ALL-IN-1 does the following:

1. Executes INPUT_WANTED with a DO function call. This means that any functions or directives in INPUT_WANTED execute in DO mode.
2. Displays a prompt asking if you want to continue. Since you called the script containing the YESNO_PROMPT function using .FX, ALL-IN-1 returns control to you to supply the input for the prompt.
3. Resumes processing FX_EXAMPLE.SCP when you respond to the prompt. Since ALL-IN-1 is ready to accept user input again, the rest of the script is used as user input. However, since Y is not a valid menu option, this causes ALL-IN-1 to display the following error message:

```
Invalid choice - re-enter
```


APPLICATION PROGRAMMING

Additional Information

For more information on scripts and script processing, refer to the chapter on scripts in the *ALL-IN-1 Application Programming: Guide*. For more information on script directives, refer to the chapter on script directives in the *ALL-IN-1 Application Programming: Reference Volume 1*.

Phantom Data Sets

This article explains what phantom data sets are and how they differ from other data sets. It demonstrates a restriction on using phantom data sets, describing an example situation where you can use a normal data set, but not a phantom data set.

A phantom data set provides a way of accessing a subset of an ALL-IN-1 data set. If you want to access a certain part of a large data set more than once, it is quicker to create a phantom data set instead of accessing the whole data set each time. Phantom data sets are also used to display records in the scrolled regions of index forms.

The BIND and BINDW functions create phantom data sets. These functions are described in *Application Programming: Reference Volume 2*. For example, the following creates a phantom data set *HAVEREAD for accessing the documents in the READ folder of the File Cabinet:

```
BIND *HAVEREAD TO CAB$ WITH .FOLDER EQS "READ"
```

In this example, the BIND function builds a list of pointers to all the records that have READ in the Folder field.

Phantom data sets are held as a list of record pointers in memory. Each pointer is a data structure that holds the key, %KEY, and record file address (RFA), %ADDRESS, for the corresponding record. Phantom data sets do not store the records themselves.

Restriction on Using Different Keys of Reference Usually, you can use a phantom data set in exactly the same way as any other ALL-IN-1 data set. But, because the phantom data set does not contain complete copies of the records in the data set, you can only access a phantom data set using the key of reference specified in the original BIND statement.

For example, consider the following data set. It is called ENTRY and has a primary key of reference KEY1 and a secondary key of reference KEY2.

ENTRY	
KEY1	KEY2
1	E
2	D
3	C
4	B
5	A

The following script displays the records in the ENTRY data set with KEY1 greater than 2, sorted in alphabetical order using KEY2:

```
FORM AUTO SELECT FOR ENTRY:KEY2 WITH .KEY1 GTS "2" -
DO SEL_DISPLAY .KEY1 " " " .KEY2
```

displays this result:

ENTRY	
KEY1	KEY2
5	A
4	B
3	C

APPLICATION PROGRAMMING

You can use a BIND statement to create a phantom data set of the records with KEY1 greater than 2. But, if you do this, you cannot sort the records in the phantom data set using the secondary key of reference. For example, the following script:

```
BIND *PHANTOM TO ENTRY WITH .%KEY GTS "2"
FORM AUTO SELECT FOR *PHANTOM:KEY2 -
DO SEL_DISPLAY .KEY1 "      " .KEY2
```

displays this result:

ENTRY	
KEY1	KEY2
3	C
4	B
5	A

The list is not in alphabetical order because you cannot access *PHANTOM using KEY2. So, the above FORM function is actually processed as the following:

```
FORM AUTO SELECT FOR *PHANTOM DO SEL_DISPLAY .KEY1 "      " .KEY2
```

The phantom data set *PHANTOM contains only the primary key and a pointer to each record. Once a phantom data set is created, you can only access its records in the order determined by the original BIND statement.

Qualifiers for the BIND Function

You can create a phantom data set with the records sorted into alphabetical order based on the value of any field in the data set. There are two qualifiers for the BIND function, /SORT and /DESCENDING. The /SORT qualifier sorts the phantom data set in the order you specify. The field that you use to sort the data set does not have to be a key of reference.

The following example creates and displays a phantom data set, sorted by KEY2, containing the records with KEY1 greater than 2:

```
BIND/SORT=".KEY2" *SORTED TO ENTRY WITH .%KEY GTS "2"
FORM AUTO SELECT FOR *SORTED -
DO SEL_DISPLAY .KEY1 "      " .KEY2
```

displays this result:

ENTRY	
KEY1	KEY2
5	A
4	B
3	C

The BIND/SORT statement in the example is equivalent to:

```
BIND *SORTED TO ENTRY:KEY2 WITH .KEY1 GTS "2"
```

BIND/SORT is more flexible because the field used for sorting does not have to be a key of reference.

You can use the second qualifier, /DESCENDING, with /SORT to create the phantom data set with the records sorted in reverse order. For example, the following creates a phantom data set *PHANTOM of documents in the File Cabinet, sorted in reverse alphabetical order by the Author field:

```
    BIND/SORT=".AUTHOR"/DESCENDING *PHANTOM TO CAB$
```

Notice that AUTHOR is not a key of reference to the File Cabinet data set.

Cumulative Index

The index on the following pages contains entries from this issue of the *ALL-IN-1 Information Update*. For each entry, the issue date and page number are shown.

INDEX

A

Address list
 editing part way through a message, *May 1992*, 8
Alias file, *May 1992*, 32
ALL-IN-1 system
 backing up, *May 1992*, 23
 ensuring integrity, *May 1992*, 23
 monitoring, *May 1992*, 11
AND operator
 See Operator
Application
 compute-intensive
 managing, *May 1992*, 28
 problems, *May 1992*, 28
 solutions, *May 1992*, 28
ASSETS package, *May 1992*, 11
ATTACH function
 See Function

B

BEGINNING operator
 See Operator
Boolean expression, *May 1992*, 55

C

CAL\$SET_DATE DSAB
 See DSAB
CAL\$TIME_CHECK DSAB
 See DSAB
CDA_CLEANUP function
 See Function
CDA_CONVERT function
 See Function
CDA_PACK function
 See Function
CDA_UNPACK function
 See Function
CM site text library (CMTXL)
 location of files, *May 1992*, 48
Command
 OPEN, *May 1992*, 18
 SHOW DEVICE, *May 1992*, 19

Compile and install CM text library (CCM)
 option
 See Menu option
Compound document
 converting format of, *May 1992*, 61
 external reference, *May 1992*, 57
 format of, *May 1992*, 57
 packing into DOTS format, *May 1992*, 63
 storage format of, *May 1992*, 57
 stored semantics attribute, *May 1992*, 57
COMPUTE function
 See Function
Compute-intensive application
 See Application
CONTAINING operator
 See Operator

D

Data
 accessing, *May 1992*, 53
Data bucket, *May 1992*, 29
 optimum size, *May 1992*, 29
Data file
 OA\$DAF_%.DAT, *May 1992*, 20
DATA_FILE subfunction
 See Subfunction
Date field, *May 1992*, 39
 comparing, *May 1992*, 40
 creating, *May 1992*, 39
 NBS format, *May 1992*, 41
 validating, *May 1992*, 39
Date format, *May 1992*, 40
 NBS, *May 1992*, 41
DATE_CONVERT function
 See Function
DDIF (Digital Document Interchange Format),
 May 1992, 57
DECLARE_METER function
 See Function
DECnet, *May 1992*, 14
 object 22, *May 1992*, 17, 18
DECREMENT function
 See Function

DECtrace for VMS, *May 1992*, 50

Delete all (XD) option
 See Menu option

Development form library (DEVELOP.FLB),
 May 1992, 27

Directory
 checking the number of files in each, *May 1992*, 20
 OA\$DATA_LLV, *May 1992*, 29
 OA\$DATA_SHARE, *May 1992*, 29
 OA\$LIB_SHARE, *May 1992*, 29

Directory Service, *May 1992*, 31
 setting up entries, *May 1992*, 32

Disk
 RA90, *May 1992*, 20

Disk space, *May 1992*, 7

Distribution list, *May 1992*, 8
 omitting when printing mail, *May 1992*, 3
 skipping when reading mail, *May 1992*, 3

Documentation
 ALL-IN-1 Application Programming: Reference, Volume 2, *May 1992*, 50
 ALL-IN-1 Application Programming: Reference Volume 1, *May 1992*, 48
 ALL-IN-1 Mail Management Guide, *May 1992*, 17
 ALL-IN-1 Management Guide, *May 1992*, 20, 50
 ALL-IN-1 Users' Reference, *May 1992*, 7
 Application Programming: Guide, *May 1992*, 46
 Guide to VMS File Applications, *May 1992*, 29
 Message Router Management Action Procedures, *May 1992*, 18
 Message Router Management Guide, *May 1992*, 19

DO function
 See Function

DO script
 See Script

DOTS (Data Object Transport Syntax), *May 1992*, 57

DSAB (data set access block), *May 1992*, 55
 CAL\$SET_DATE, *May 1992*, 39
 CAL\$TIME_CHECK, *May 1992*, 43

DSR (Data Set Reference), *May 1992*, 53
 using, *May 1992*, 53

DTIF (Digital Table Interchange Format),
 May 1992, 57

E

Elapsed time
 calculating, *May 1992*, 50

Electronic messaging
 checking who received, *May 1992*, 8
 clearing out old messages, *May 1992*, 4
 monitoring, *May 1992*, 11, 13
 shared area, *May 1992*, 7
 skipping long distribution lists, *May 1992*, 3

Electronic Messaging (EM) menu, *May 1992*, 4

Element Speed Measurement (ESM) option
 See Menu option

ENS operator
 See Operator

EQS operator
 See Operator

External reference, *May 1992*, 57

F

FDL (File Definition Language), *May 1992*, 29, 54

FETCHCHECK program
 See Program

Fetcher queue
 See Queue

Field validation, *May 1992*, 55

File
 checking the number in each directory, *May 1992*, 20

File Attachment (FA) option
 See Menu option

File Cabinet
 repairing, *May 1992*, 23
 testing, *May 1992*, 23

File protection
 checking major files, *May 1992*, 21

File Text (FT) option
 See Menu option

FMS
 avoiding special features, *May 1992*, 47
 Field attribute
 Right Justify, *May 1992*, 35
 Zero Fill, *May 1992*, 35

FMS form driver
 See Form driver

Folder
 outbox, *May 1992*, 4
 read, *May 1992*, 4

FOR function
 See Function

Form

- designing, *May 1992*, 47
- Editor Attributes (EA), *May 1992*, 3
- FMS, *May 1992*, 45
- FORM function, *May 1992*, 46
- libraries, *May 1992*, 45
- number limit, *May 1992*, 46
- optimum value, *May 1992*, 47
- options, *May 1992*, 46
- performing an operation without calling a,
May 1992, 47
- size, *May 1992*, 45

Form directory

- See* Directory

Form driver

- FMS, *May 1992*, 27

FORM function

- See* Function

Form library, *May 1992*, 27

- DEVELOP.FLB, *May 1992*, 27

Function

- ATTACH, *May 1992*, 50
- BOTTOM, *May 1992*, 4
- CDA_CLEANUP, *May 1992*, 59
- CDA_CONVERT, *May 1992*, 61
- CDA_PACK, *May 1992*, 63
- CDA_UNPACK, *May 1992*, 65
- COMPUTE, *May 1992*, 48
- COPY, *May 1992*, 50
- DATE_CONVERT, *May 1992*, 40
- DECLARE_METER, *May 1992*, 50
- DECREMENT, *May 1992*, 48
- DELETE_FILE, *May 1992*, 50
- DO, *May 1992*, 48
- EDIT, *May 1992*, 50
- EXECUTE, *May 1992*, 50
- FOR, *May 1992*, 48
- FORM, *May 1992*, 46
- GET, *May 1992*, 3, 47
- GET_RMS_SEMANTIC_TAG, *May 1992*,
69
- GOLD GET, *May 1992*, 3
- INCREMENT, *May 1992*, 48
- INSTALL, *May 1992*, 50
- MERGE, *May 1992*, 28, 48
- NEWDIR, *May 1992*, 25
- OA\$LOG_TO_FILE, *May 1992*, 50
- OA\$STAT_GET_STATISTICS, *May 1992*,
50
- PURGE_FILE, *May 1992*, 50
- QUEUE_BATCH, *May 1992*, 50
- RENAME, *May 1992*, 50
- SCRIPT, *May 1992*, 48
- SELECT, *May 1992*, 4
- SPAWN, *May 1992*, 50
- XOP, *May 1992*, 48

.FUNCTION script directive

- See* Script directive

.FX script directive

- See* Script directive

G

GET DOCMT screen on field, *May 1992*, 3

GET function

- See* Function

GET_RMS_SEMANTIC_TAG function

- See* Function

Global buffer, *May 1992*, 29

GOTO script directive

- See* Script directive

H

Housekeeping procedures

- Test and Repair Mail Areas (TRM), *May 1992*, 23

- Test and Repair User Areas (TRU), *May 1992*, 23

I

INCREMENT function

- See* Function

Indexed search, *May 1992*, 53, 55

K

Key of reference, *May 1992*, 53

L

.LABEL script directive

- See* Script directive

Live link

- See* External reference

Logical

- checking, *May 1992*, 21

- MR\$NODE, *May 1992*, 18

- OA\$MTI_MAILBX, *May 1992*, 14

- OA\$MTI_MR_NODE, *May 1992*, 14, 18

- OA\$SHAR*, *May 1992*, 19

M

Mail

- See* Electronic messaging

Menu option

- Compile and install CM text library (CCM),
May 1992, 28

- Delete all (XD), *May 1992*, 4

- Element Speed Measurement (ESM), *May 1992*, 50

- File Attachment (FA), *May 1992*, 3

- File Text (FT), *May 1992*, 3

Menu option (cont'd)

Precompile a form library (PCF), *May 1992*, 27

Print (PO), *May 1992*, 5

Refile document (RFD), *May 1992*, 4

Search mail directory (SMD), *May 1992*, 31

User setup (US), *May 1992*, 5

MERGE function

See Function

Message

deferred

how to check, *May 1992*, 13

on the Sender queue, *May 1992*, 13

Message Router

checking availability, *May 1992*, 18

on a remote node

how to check, *May 1992*, 18

Message Router ALL-IN-1 mailbox, *May 1992*, 14

how to check, *May 1992*, 14

Message Router Directory Service

See Directory Service

Metering

using, *May 1992*, 50, 51

MRP (Message Router/P Gateway), *May 1992*, 31

MRS (Message Router/S Gateway), *May 1992*, 31

MR-Telex (Message Router Telex Gateway), *May 1992*, 31

MRX (Message Router X.400 Gateway), *May 1992*, 31, 32

MTA (Message Transfer Agent), *May 1992*, 32

N

Named data, *May 1992*, 48

EM menu, *May 1992*, 47

O

OA\$DATA_LLV directory

See Directory

OA\$DATA_SHARE directory

See Directory

OA\$FILE_SEARCH_ORDER symbol

See Symbol

OA\$LIB_SHARE directory

See Directory

OA\$STATUS

checking value, *May 1992*, 48

OA\$TRA_SET IO, *May 1992*, 55

OA\$TXL_SEARCH_ORDER symbol

See Symbol

OPEN command

See Command

Operation

GET_BY_KEY, *May 1992*, 55

GET_NEXT, *May 1992*, 55

LOCATE, *May 1992*, 55

Operator

AND, *May 1992*, 54

BEGINNING, *May 1992*, 54, 55

CONTAINING, *May 1992*, 55

ENS, *May 1992*, 54, 55

EQS, *May 1992*, 54, 55

P

Performance

cost, *May 1992*, 45

designing an application for, *May 1992*, 4

maintaining files for better, *May 1992*, 2

managing an application for, *May 1992*, 1

perceived, *May 1992*, 45

programming an application for, *May 1992*, 45

speed of, *May 1992*, 27

testing applications for, *May 1992*, 50

Permanent symbol table file (.PST), *May 1992*, 49

Precompile a form library (PCF) option

See Menu option

Print (PO) option

See Menu option

Printing

mail

omitting long distribution lists, *May 1992*, 3

saving paper, *May 1992*, 5

Private document, *May 1992*, 7

Program

FETCHCHECK, *May 1992*, 14, 15

SENDCHECK, *May 1992*, 13

Q

Qualifier

/NOREENTER, *May 1992*, 25

/REENTER, *May 1992*, 25

/RSE_VALID, *May 1992*, 55

/VALID, *May 1992*, 55

Queue

Fetcher, *May 1992*, 13

how to check, *May 1992*, 14

monitoring the size, *May 1992*, 14

Sender, *May 1992*, 13

deferred messages, *May 1992*, 13

R

RA90 disk
 See Disk
Refile document (RFD) option
 See Menu option
RMS (Record Management Services), *May 1992*, 53
RSE (record selection expression)
 rules, *May 1992*, 54
 using, *May 1992*, 54
 with the FOR function, *May 1992*, 53
 writing, *May 1992*, 54

S

Script
 DO, *May 1992*, 28, 47, 48
 DO mode, *May 1992*, 71
 nesting, *May 1992*, 73
 SCRIPT, *May 1992*, 28, 48
 SCRIPT mode, *May 1992*, 71
 writing guidelines, *May 1992*, 47
Script directive
 .FUNCTION, *May 1992*, 71
 .FX, *May 1992*, 71
 .GOTO, *May 1992*, 47
 .LABEL, *May 1992*, 47
SCRIPT function
 See Function
Scrolling
 switching off during GOLD GET, *May 1992*, 3
SDAF (Shared Document Attributes File),
 May 1992, 23
Semantic tag
 defined, *May 1992*, 57
SENDCHECK program
 See Program
Sender queue
 See Queue
Sequential search, *May 1992*, 53, 55
Shadowed disk, *May 1992*, 24
Shared Document Attributes File
 See SDAF
Site text library
 CM, *May 1992*, 48
SPAWN function
 See Function
Special field
 %CLOSE, *May 1992*, 53, 54
 %KEY, *May 1992*, 54
 %OPEN, *May 1992*, 53, 54
Stored semantics attribute, *May 1992*, 57
Subfunction
 DATA_FILE, *May 1992*, 53, 55

Subprocess

 avoiding use of, *May 1992*, 49

Symbol

 adding leading zeros to, *May 1992*, 35
 comparing, *May 1992*, 49
 OA\$DATE_NBS, *May 1992*, 50
 OA\$DA_FORM_CACHE_SIZE, *May 1992*, 47
 OA\$FILE_SEARCH_ORDER, *May 1992*, 48
 OA\$MENU_REMAINDER, *May 1992*, 47
 OA\$PROFIL_DIRECT, *May 1992*, 49
 OA\$TXL_SEARCH_ORDER, *May 1992*, 48
 right-justifying, *May 1992*, 35
 special, *May 1992*, 49
 specifying substrings, *May 1992*, 35
 using, *May 1992*, 49

System monitoring

See ALL-IN-1 system

T

Test and Repair Mail Areas (TRM)

See Housekeeping procedures

Test and Repair User Areas (TRU)

See Housekeeping procedures

Text library, *May 1992*, 27

 compiling, *May 1992*, 28
 installing, *May 1992*, 28
 managing, *May 1992*, 27
 noncompiled, *May 1992*, 27
 precompiled, *May 1992*, 27

Time field, *May 1992*, 39, 42

 creating, *May 1992*, 42
 validating, *May 1992*, 42

Trace facility, *May 1992*, 50

U

UDP (user-defined process), *May 1992*, 7

 checking who received mail, *May 1992*, 8
 counting

 number of lines in document, *May 1992*, 9

 number of pages in document, *May 1992*, 9

 number of paragraphs in document, *May 1992*, 9

 number of sentences in document, *May 1992*, 9

 number of words in document, *May 1992*, 9

 creating, *May 1992*, 7

 using, *May 1992*, 7

UMC (ULTRIX Mail Connection), *May 1992*, 31, 32

User

hints for, *May 1992*, 3

User-Defined Process Maintenance menu,
May 1992, 7

User setup (US) option

See Menu option

V

VAXcluster system, *May 1992*, 21

W

Wastebasket, *May 1992*, 4

WPS-PLUS

incorporating text from one document into
another, *May 1992*, 3

X

X.400

See MRX

XOP function

See Function

Attention U.S. Software Contract Customers Submitting Software Problems

As part of Digital's continuing effort to improve customer satisfaction, U.S. Software Product Services (SPS) has changed the process for U.S. customers to submit software problems. These customers should report their software problems by calling their Customer Support Center (CSC) or by accessing either the Digital Software Information Network (DSIN) or DSNlink.

The CSCs have the technical expertise to quickly respond to many of our customers' problems without requiring Engineering intervention. Because many software problems are duplicates of problems that have already been addressed by Engineering, the CSCs can immediately respond to many inquiries.

If you do not have access to a CSC, submit your software problem to:

**Digital Equipment Corporation
P.O. Box 1167
Alpharetta, Georgia 30239-1167**

This address replaces the one currently found on the instruction sheet of the Software Performance Report (SPR) form.

Digital Equipment Computer Users Society

Benefits of Belonging

The Digital Equipment Computer Users Society (DECUS) is one of the largest and most respected users groups in the computer industry today. Membership in DECUS, which is free and voluntary, affords the individual user information and services not found anywhere else.

DECUS provides an environment where users of Digital Equipment Corporation products can share information with other users and with Digital. Members can find out the latest news on Digital's hardware, software, and educational products. The feedback exchange with Digital allows the users of Digital's products a voice in the direction the company takes in the future.

Founded in 1961, DECUS has three autonomous chapters worldwide—DECUS U.S., DECUS Europe, and DECUS General International Area (GIA). DECUS services and activities are shared between these chapters through mutual agreements.

All DECUS services promote the exchange of information in a commercial-free environment. These services include:

Special Interest Groups (SIGs)

These groups, formed around an area of common interest, exist for a variety of hardware, operating systems, languages, applications, and marketing areas. Participation in these groups allows contact among fellow users to exchange information and share technical expertise in the area of most interest to the user.

Local Users Groups (LUGs) and National Users Groups (NUGs)

LUGs and NUGs are licensed groups of individuals who gather to share information with other users on a periodic basis. Not only do these people share professional interest, but they also share geographic and cultural ties. Often, Digital representatives attend these meetings to discuss new products and services and supply updates on existing policies and procedures.

Symposia

DECUS holds symposia each year in the different chapters, two per year in the U.S. These meetings provide unique opportunities for users with a wide spectrum of experience to meet for up to five intensive days of technical exchange. Included in symposia activities are workshops, clinics, panels, tutorials, and formal paper presentations. Digital participates in symposia by sending product group managers and developers to discuss strategies, products, problems, and solutions.

Communications

The flow of information among users, as well as between users and Digital, is the primary goal of DECUS. Various publications are published by DECUS to support this communication. They include chapter newsletters and *Transactions*, a technical volume published after each symposium.

DECUS also publishes the Special Interest Groups *SIGs Newsletter* on a monthly basis. The newsletter is divided into Special Interest Group sections and provides specialized information pertaining to specific Digital products. The monthly newsletter and *Transactions* are available through the DECUS Subscription Service.

Seminars

DECUS recognizes the need for ongoing training opportunities that help members keep pace with the changing technologies of the computer industry. To that end, DECUS sponsors technical seminars that run from 1/2 day to 2 days throughout the year in various locations. These seminars are developed and presented by DECUS members who are experienced industry professionals.

Program Library

The DECUS Program Library is the main vehicle for the exchange of public domain software among users of all Digital systems. The library contains over 1000 software programs written and voluntarily submitted by users. These programs include compilers, editors, utilities, numerical and statistical functions, as well as games and graphic routines. The library publishes a software catalog on an annual basis that lists and describes all the DECUS programs.

DECUServe

DECUServe is an electronic conferencing system based on VAX Notes. The network is available to all U.S. and Canadian members 24 hours a day, 7 days a week for an annual subscription fee.

Joining DECUS

You are cordially invited to join with over 100,000 other users of Digital products around the world and begin to share your experiences, both the successes and problems.

For more information, contact the appropriate DECUS Chapter Office.

DECUS Chapter Offices—Worldwide

DECUS U.S.

DECUS U.S.
333 South Street, (SHR1-4/D30)
Shrewsbury, Massachusetts 01545-4195
Tel. (508) 841-3389

DECUS Europe

DECUS Europe Headquarters
12 Avenue des Morgines
Case Postale 176
CH-1213 Petit-Lancy 1/Geneva
Switzerland

DECUS BELUX
Luchtschipstraat 1
Rue de l'Aeronef
B01140 Brussel/Bruxelles
Belgium

DECUS Finland
P.O. Box 6
SF-02201 Espoo
Finland

DECUS Holland
Europalaan 44
P.O. Box 9212
3506 GE Utrecht
The Netherlands

DECUS Ireland
c/o Digital Equipment Ireland Ltd.
Park House, North Circular Road
IRL-Dublin 7
Ireland

DECUS Italia
Vie F. Testi, 280/6
I-20126 Milano-MI
Italy

DECUS At-Large Chapter
12 Avenue des Morgines
Case Postale 176
CH-1213 Petit-Lancy 1/Geneva
Switzerland

DECUS Denmark
c/o Digital Equipment Corporation A/S
Aadalsvej 99
DK-2970 Hoersholm
Denmark

DECUS France
2 Rue Gaston Crenieux
BP136
91004 Evry Cedex
France

DECUS Iceland
c/o DECUS Denmark

DECUS Israel
c/o Digital Equipment
Acadia Junction
Herzlia 46 733
Israel

DECUS Muenchen
Freischuetzstrasse 91, Postfach 810247
W-8000 Muenchen 81
West Germany

DECUS Chapter Offices—Worldwide (Continued)

DECUS Norway
Ammerudveien 22
N-0958 Oslo 9
Norway

DECUS Portugal
c/o Digital Equipment Portugal Lda.
Empreendimento Torres das Amoreiras
Av. Eng. Duarte Pacheco, Torre 1-9 andar
P-1000 Lisbon
Portugal

DECUS Spain
c/o Digital Equipment Corporation SA
Cerro del Castanar 72, Mirasierra
28034 Madrid
Spain

DECUS Sweden
Gardavagen 1
S-402 26 Gothenburg
Sweden

DECUS Switzerland
DECpark
Ueberlandstrasse 1 Postfach
CH-8600 Dubendorf 1
Switzerland

DECUS U.K., Ireland, and Middle East
Queen's House
Forbury Road
GB-Reading, RG 1 3JJ
U.K.

DECUS GIA (General International Area)

DECUS GIA Headquarters
100 Nagog Park AKO1-1/B11
Acton, Massachusetts 01720
U.S.A.

DECUS South Pacific Chapter
Australia Office
Locked Bag 16, 410 Concord Road
Rhodes, New South Wales 2138
Australia

DECUS
New Zealand Office
P.O. Box 8610
Symonds Street
Auckland 3, New Zealand

DECUS Canada
505 University Avenue, 15th Floor
Toronto, Ontario M5G 1X4
Canada

DECUS Far East
Digital Equipment Corporation
19-21 Fleet House
38 Gloucester Road
Wanchai, Hong Kong

DECUS Japan
Nihon Digital Equipment KK
Sunshine 60, P.O. Box 1135
1-1. Higashi Ikebukuro 3 - chome
Toshima-ku Tokyo
Japan 170

DECUS South America
Digital Equipment do Brasil Ltda.
Av. Presidente Wilson
231 26th Floor
20030 Rio de Janeiro, RJ
Brasil

DECUS Latin America/Carribean Chapter
DECUS LACC
800 Fairway Drive
Suite 400
Deerfield Beach, FL 33441
U.S.A.

DECUS SUBSCRIPTION SERVICE
SIGs NEWSLETTER
U.S. Chapter Transactions
(U.S. Chapter Members Only)

As a member of the DECUS U.S. Chapter, you are entitled to contribute and subscribe to the DECUS monthly publication, *SIGs Newsletter*. You also have the opportunity to subscribe to Transactions which are a compendium of the reports from various speakers at the U.S. National DECUS Symposia.

- The order form below must be used as an invoice
- All checks must be made payable to DECUS
- All orders **must** be paid in full
- \$25.00 minimum for credit card orders
- No refunds will be made
- The address provided below will be used for all DECUS mailings, e.g., membership, subscription service, and symposia
- SIGs Newsletter price is for a one-year subscription beginning the month following receipt of payment

.....

Name _____ DECUS Member No. _____

Company _____

Address _____

City _____ State _____ Zip _____

Phone _____

Subscription Service Offering	Qty	Unit Price	Total
SIGs Newsletter	_____	\$40.00	_____
Fall '90 Proceedings (FA0)	_____	15.00	_____
Spring '91 Proceedings (SP1)	_____	15.00	_____
Fall '91 Transactions (FA1)	_____	25.00	_____
Spring '92 Transactions (SP1)	_____	25.00	_____
Total Cost of Subscription			\$ _____

☐ MasterCard® ☐ VISA® ☐ DINERS CLUB® ☐ CARTE BLANCHE® ☐ American Express®

Card No. _____ Exp. Date _____

I understand that there will be no refunds, even if I decide to cancel my subscription.

Signature _____

For Digital Employee Use Only

Badge No. _____ CC: _____

CC Mgr Name _____

CC Mgr Signature _____

For DECUS Office Only

Check No. _____

Bank No. _____

Amount \$ _____

Send to: Subscription Service, DECUS, 333 South Street (SHR1-4/D30), Shrewsbury, MA 01545-4195, (508) 841-3389.

Note: DECUS Europe and DECUS GIA members should contact their local DECUS office.



SD891205
MR-4829-R/

Reader's Comments

We welcome your comments and suggestions regarding the *ALL-IN-1 Information Update*. They will help us in our continuing effort to improve the quality and usefulness of the Update. Please take a few minutes to complete this form and return it to us.

What do you think of the Update content?

- ☐ Too technical
- ☐ Not technical enough
- ☐ About right

How often do you refer to the Update?

- ☐ Occasionally
- ☐ Regularly
- ☐ Only to solve a problem

Is the Update of general use to you? (explain)

- ☐ Yes _____
- ☐ No _____

What other information would you like to see in the Update? _____

Please outline any errors you have found in the Update or any suggestions for improvement. _____

(Attach additional pages, if necessary)

Additional comments: _____

How many years experience do you have in the following?

- _____ ALL-IN-1
- _____ EDT and/or WPS editor
- _____ WPS-PLUS editor
- _____ Other office automation systems
- _____ Other word processing systems

I am/am not* willing to have you contact me by mail/telephone* regarding my comments and suggestions.

Name: _____

Title: _____

Company: _____

Street: _____

City: _____ State: _____ Zip Code: _____

Country: _____ Telephone: _____

*Delete as applicable.

Do Not Tear — Fold Here and Tape

digital™



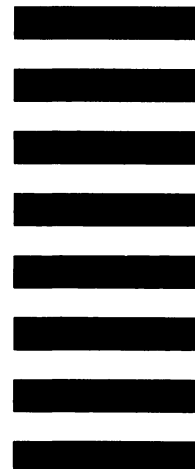
No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD, MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Susan A. Thompson
Editor, ALL-IN-1 Information Update
Digital Equipment Corporation
Corporate User Information Products (MR01-3/T2)
P. O. Box 1001
Marlborough, MA 01752-9840



Do Not Tear — Fold Here and Tape